Technical Document 2313
July 1992

Shipboard Readiness Reporting System (SRRS) Software Prototype

Maniel Vineberg
Stan Connors
Tony Sterrett
Dave Shore

Technical Document 2313
July 1992

Shipboard Readiness Reporting System (SRRS) Software Prototype

Maniel Vineberg
Stan Connors
Tony Sterrett
Dave Shore

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

The study covered in this document was performed from 1 November 1990–June 1992. It was sponsored and funded by the David Taylor Research Center, Bethesda, MD, under accession number DN301014, program element 0602233N, project no. EE03, and subproject no. RM33D61. The work was performed by the Naval Command, Control and Ocean Surveillance Center, RDT&E Division (NRaD), San Diego, California.

Released by
J. T. Avery, Head
Systems Analysis Group

Under authority of
J. T. Avery, Head (Acting)
Planning, Intelligence, and
Analysis Office

Specifically, work on the Shipboard Readiness Reporting System was performed under the following organizations:

Task — Shipboard Readiness Reporting System, EE03,
Maniel Vineberg, NRaD, Code 171

Project — Advanced Diagnostic Techniques, RM33R61,
Bill Nickerson, ADCD-NSWC, Code 272Z

Block — Logistics, ND2A,
Ray Brengs, CD-NSWC, Code 0116

Sponsor — Mission Support, PE 62233N,
CDR David Bennett, ONT, Code 225.

RBT

CONTENTS

CONTENTS (continued)

FIGURES

INTRODUCTION

OVERVIEW

This document describes a software prototype of a Shipboard Readiness Reporting System (SRRS) (reference 1). SRRS will provide useful, accurate, concise, and timely readiness status reports to users who operate and maintain shipboard systems.

SRRS is designed in the context of a layered open architecture, anticipating future shipboard architectures. The highest layer of the architecture includes both operational functions (relating directly to the ship's mission) and readiness reporting functions. SRRS will collect and synthesize periodic status reports from shipboard resources and automatically report operational capability and maintenance requirements, based on resource status, to tactical and technical users.

BACKGROUND

SRRS will ensure continuous measurement and reporting of the operational and maintenance readiness of the entire surface combatant. A ship comprises systems, and a system uses combinations of resources to perform functions. A number of shortcomings, documented in reference 1, result from a lack of resource reporting standards (reference 2) and the absence of support to automatically consolidate and present status reports.

RELATED EFFORTS

SRRS practices and communication techniques, developed for the combat system, are compatible with "Condition-Based Maintenance," developed by the Annapolis Detachment of the Carderock Division of the Naval Surface Weapons Center (ADCD-NSWC) for the hull, mechanical, and electrical (HM&E) system (reference 3).

SCOPE AND OBJECTIVES

The objective of this task is to develop an SRRS by combining data processing, communications, built-in-test (BIT), and calibration hardware and software. SRRS will provide useful, accurate, concise, and timely readiness status reports to *tactical* and *technical users* responsible for operating and maintaining shipboard systems. The reports will include the readiness data and information needed to make correct and timely operational and maintenance decisions.

While battle-force readiness reporting is an SRRS objective, FY-92 work was restricted to the individual platform. Note that each ship must have complete

knowledge of its own readiness to support battle-force readiness reporting. The SRRS focus during FY 92 was on the combat system; however, the HM&E system can be integrated in FY 93.

DESIGN PHILOSOPHY

SRRS is based on five principles. First, SRRS is primarily intended for new systems rather than for being retrofitted into existing systems. However, parts of SRRS can be used to identify current problems and to provide temporary corrections for in-service ships.

Second, SRRS specifies what must be reported; designers of shipboard equipment determine how reports are generated. This is consistent with the philosophy of open architecture.

Third, SRRS standards will be established with industry to assure viability and to promote "commercial dual use."

Fourth, to successfully assess and report status during combat, SRRS must survive damage to the resources it uses, particularly the communications media. *Graceful degradation* will be designed into SRRS to raise the probability of successfully reporting damage.

Fifth, SRRS must present the "right amount" and "right kind" of information to tactical and technical users and, upon requests, must also provide timely access to extensive detail.

APPROACH

The SRRS approach consists of a design—within the context of a layered open architecture—and a software prototype. SRRS design includes these features:

a. Interface specifications that define BIT formats and protocols for resource designers.

b. A data-distribution system that links resources.

c. A distributed operating system that provides, among other things, protocols to allow distributed resources to communicate.

d. Resource and file managers that allocate hardware and software.

e. Applications that collect, process, format, and distribute readiness data and information from resources.

f. Applications that synthesize resource readiness reports to determine operational capability (what functions can be performed) and maintenance needs (what resources need repair).

g. Functions and services SRRS provides to the user.

h. Metrics, to assess the quality and the performance of SRRS.

The software prototype is an unambiguous characterization of SRRS design intended to enable

a. Validation of the operational concept.

b. Performance measurement.

c. Demonstration and review.

d. Cost-effective revision and test.

e. Subsequent installation of SRRS on a hardware testbed.

The remainder of this document is organized into two parts: SSRS design; and, the software prototype, the primary focus of the document.

SRRS DESIGN

The architecture is described first. Then, major elements of SRRS are described within the context of the architecture.

ARCHITECTURAL CONTEXT

SRRS architecture, depicted in figure 1, is based on the Battle Management Architecture (reference 4).

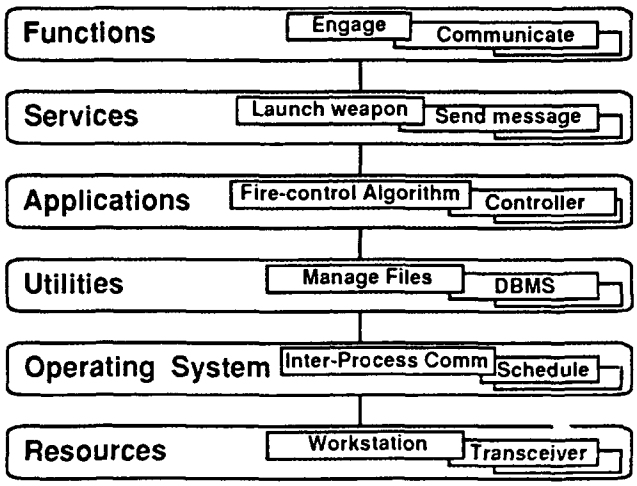


Figure 1. SRRS architecture.

Each layer in the architecture supports the next higher layer, and interfaces between layers are standardized. The layering allows one layer to be changed with minimal effect on other layers. The various layers carry the following meanings:

Functions are system operations a user may request.

Services are primitive system operations used for implementing functions.

Applications and *utilities* are special-purpose and general-purpose software processes, respectively, that support the services.

The operating system is software that makes the resources usable.

Resources, including both special-purpose (weapons, sensors, etc.) and general-purpose (workstations, networks, etc) hardware, are the hardware devices that host the applications and utilities.

The architecture enables readiness functions to be defined that portray operational capability with respect to the sets of operational functions and services the system can—or cannot—perform with the resources available.

SRRS FUNCTIONS

A function is an operation a user may request of the system, as shown in figure 2. To each request, the user assigns a unique priority, carried by the system throughout all operations related to that function and used to resolve contention for resources.

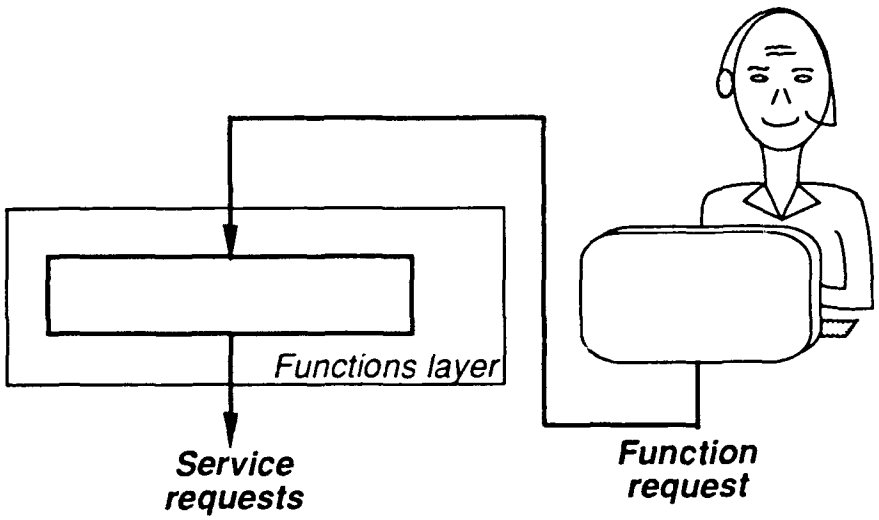


Figure 2. Functions layer.

SRRS perceives three classes of functions:

administrative functions that allow users to configure the system;

operational functions that directly support the mission of the ship; and

readiness functions that support assessment of operational capability and maintenance requirements. Currently defined are one readiness function, *report readiness*, and two operational functions, *locate submarines* and *determine ownship course and speed.*

A controller within the functions layer accepts user function requests and translates these into one or more service requests that it passes to the services layer. If that layer cannot ensure that it can perform all requested services (e.g., for lack of available resources), the controller either cancels or delays the function (according to user preference). This satisfies an underlying design rule that no single function request can cause system deadlock. This rule, together with a mechanism to ensure that every active function carries a unique priority, will prevent system deadlock. Canceled functions may be resubmitted at a higher priority.

SRRS SERVICES

Services are system primitives that implement functions. In general, several services implement a function; one service may be used to implement more than one function. Services are grouped into *five elements* as shown in figure 3. The *decision support element* includes services related to command planning and decision making through assessment and evaluation such as decision aids, tracking algorithms, planning aids, etc. Readiness reporting falls within the decision-support element: two services have been defined, *report operational capability* and *report maintenance requirements.*

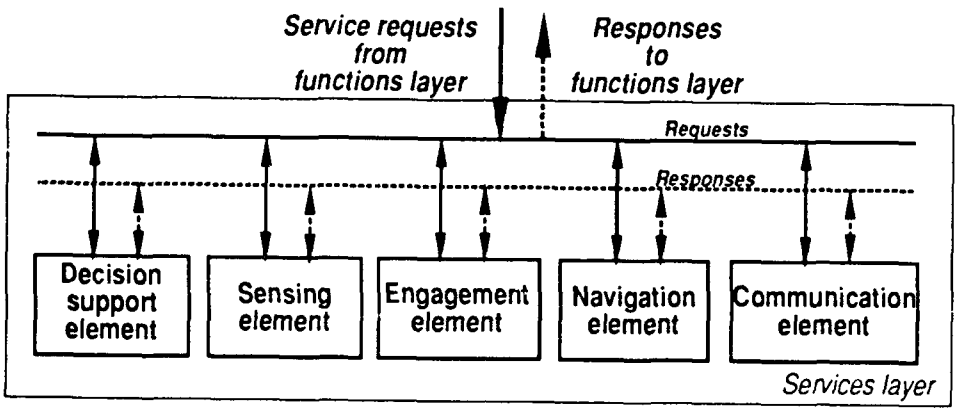


Figure 3. Services layer.

The *sensing element* includes services to detect and characterize objects and to characterize the environment external to the node. Services defined for the sensing element that supports the function *find submarines* are *deploy sonobuoys* and *process sonobuoy data*. Appendix A shows *dependency trees* illustrating resources that can perform these services.

The *navigation element* includes services to (1) supply navigation information (maps, tracks, etc.), (2) permit all navigation modifications, and (3) support mode switching (from manual mode to automatic and vice versa). Services defined for the navigation element that support the function *report current platform position and velocity* are *determine platform course and position* and *determine platform speed;* Appendix A shows dependency trees that illustrate combinations of resources that can perform these services.

The *engagement element* includes services for neutralizing external threats to the platform. The *communication element* provides interplatform communication at force level (intraplatform communication is provided by the platform data-distribution system and the operating system).

A controller within each element handles the service requests. To perform a service, a controller may issue requests to other controllers. Element controllers must be active at all times during system operation to accept and process requests for service. A controller determines if a service request can be honored; included in that determination is whether supporting service requests can be honored. Once it has been determined that all services required to implement a function can be performed, element controllers gain control of the needed resources. Elements retain control of those resources during the service session (as long as needed) and relinquish control afterward.

SRRS APPLICATIONS

Applications, portrayed in figure 4, are software processes that support services. Several applications may be required to support a single service. Readiness applications, discussed below, must be active at all times; an approach to increasing process survivability is discussed under utilities later in this document.

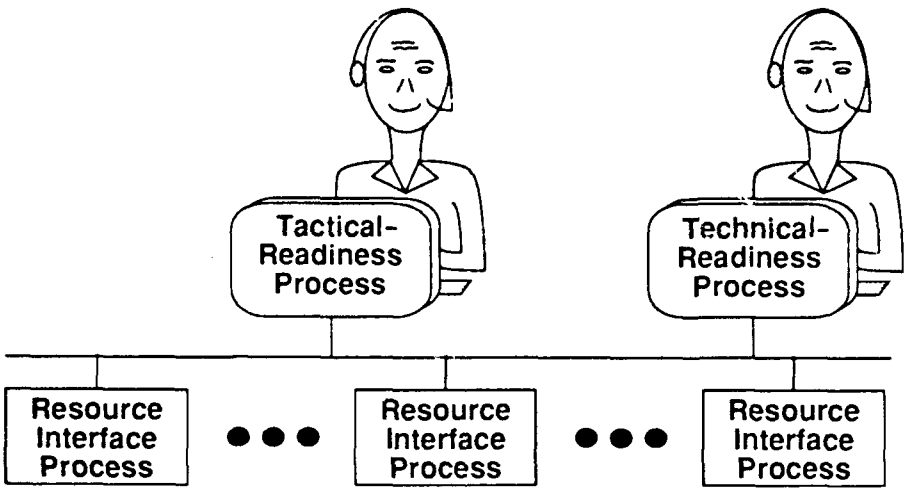


Figure 4. Applications layer.

Resource Interface Process (RIP)

Resource Interface Processes (RIPs) comprise a class of readiness applications, one for each resource. A RIP accepts the results of tests within the resource and reports them (in a standard way to be established) to the Tactical Readiness Process (TaRP) and the Technical Readiness Process (TeRP), discussed below.

Typically, a resource status report will include the following:

1. *Resource name* – each report is for a single resource. Synthesis will be done by the TaRP and the TeRP (see below).

2. *Capability* - the basic categories are *up* and *down*, however, partial capabilities will be reported concerning support of services; i.e., completely, partly, or not at all. Expected times to recover to *up* are included for resources not yet *up*.

3. *Failures* - the basic categories are *none* and *total*. The categories *redundant* and *nonredundant* are used to report failures that do not cause capability to be reported as *none*. If not *none*, expected times to repair are included.

4. *Initialization* - categories are *on*, *standby*, *energized*, *support*, and *off*. If not already *on*, expected time to *on* is included.

5. *Configuration* - categories are *normal*, *alternate*, *casualty*, or *unavailable*. If not already *normal*, expected time to *normal* is included.

6. *Remarks*.

Tactical Readiness Process (TaRP)

The *TaRP* is a readiness application that synthesizes reports from RIPs to determine operational status. Operational status is characterized by what services and functions the system is capable of performing, separately or in combination. This information is directly available to the tactical user who may use it to establish priorities for repairing resources or changing the system configuration.

In addition to resource status reports, inputs to the TaRP generally include the following: (1) a correlation between services and required resources; (2) proposed changes to resources or services; and (3) the present configuration of active services.

The TaRP supports these services: (1) "report system status," (2) "assess impact of resource change," and (3) "recommend resource change to restore service." For the three services, the TaRP produces the following: for 1 and 2, reports of services that cannot be performed at all, that cannot be performed if other services are ongoing, and that cannot be initiated under the present configuration of services; for 3, alternative modifications to resources to restore requested service.

Technical-Readiness Process (TeRP)

The *TeRP* is a readiness application that uses reports from the RIPs and priorities from the TaRP to provide reports to technical users to support scheduling of maintenance actions. The technical user may initiate repair actions in response to faults—with respect to priority (as determined by tactical users) and expected time to repair. Some repairs may be in collocated cabinets and may therefore be scheduled on the basis of convenience rather than operational priority. Others may be less urgent because they are in cabinets protected by fault tolerance. The outcome of all repair

actions are reported to the TeRP; i.e., successful, incomplete (and expected time required to complete) or unsuccessful (and reason, e.g., requiring external support). The TeRP must accept "special conditions," such as sea state or emission control, that may influence tolerances or use.

SRRS UTILITIES

Utilities (figure 5) are general-purpose processes that support applications. A *Database Manager* maintains user profiles and mailboxes; eventually, the Database Manager will manage application-specific data such as tracks. A *Resource Manager* determines resource availability, with respect to specified services, and allocates resources on demand.

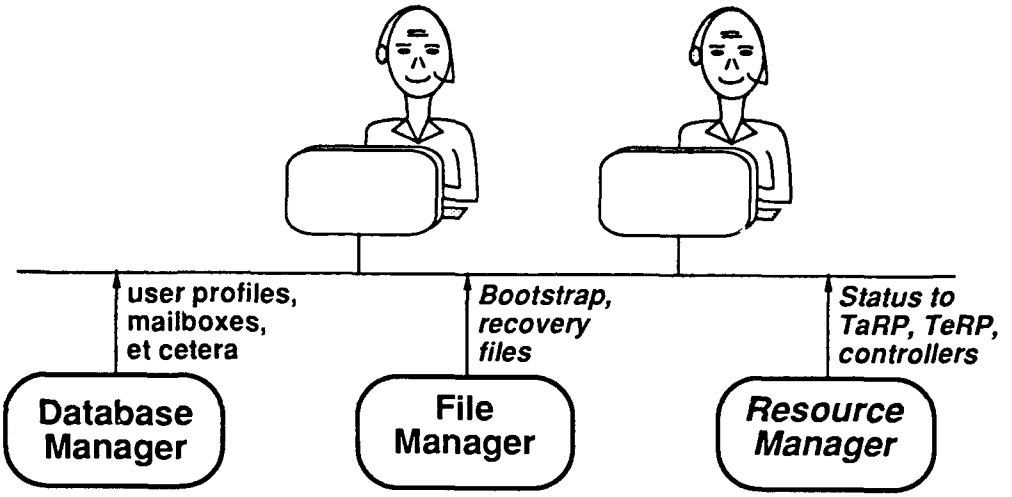


Figure 5. Utilities layer.

A *File Manager* distributes all applications and utilities to where they are needed. To initiate a system bootstrap, a copy of the File Manager is loaded externally. This File Manager controls the loading of all utilities, controllers, and readiness applications; it loads a *primary* and a *backup* copy of all such processes to support eventual recovery from loss. Copies communicate with each other periodically, via "I am OK" reports. A copy that has not heard from its mate for a given interval assumes a failure, takes the role of primary copy, and requests the *File Manager* to install a new backup copy. This approach to recovery is open to study and refinement.

SRRS OPERATING SYSTEM

The operating system is the software that makes resources usable by supporting the following tasks:

Scheduling intraresource operations;
Allocating and deallocating low-level resources (peripherals);

Allocating and deallocating main, secondary, and archival memory;
Managing interprocess communication using LAN protocols;
Managing intraplatform circuit switching using LAN protocols;
Monitoring and measuring performance;

Maintaining logs, system state, and backup information;
Managing system initialization and recover (warm and cold system start);

Managing intraplatform electronic mail;
Maintaining user profiles;
Maintaining user mailboxes and supplying them when and where needed;
Initiating user sessions using login procedures, account and password verification, and user profiles;
Terminating user sessions using logout procedures, saving files and any user-profile information.

The Next Generation Computer Resources (NGCR) program has identified the Portable Operating System Interface for Computer Environments (POSIX) (reference 5), as a Navy standard. No SRRS development is anticipated in this area.

SRRS RESOURCES

Many devices now common in the commercial sector will be useful in constructing Navy systems. Extensive use of these general-purpose resources will (1) facilitate building systems, (2) simplify special-purpose resources, and (3) support evolution (as the commercial sector continues to provide more capable versions of these devices). Several of these resources—LANs, workstations, mass memories—make SRRS feasible. A sample connection scheme is illustrated in figure 6. For example, the LAN is not only the means by which a distributed system is linked; it is the mechanism by which status reports are sent from resources (RIPs) and collected for synthesis (at a TaRP or a TeRP).

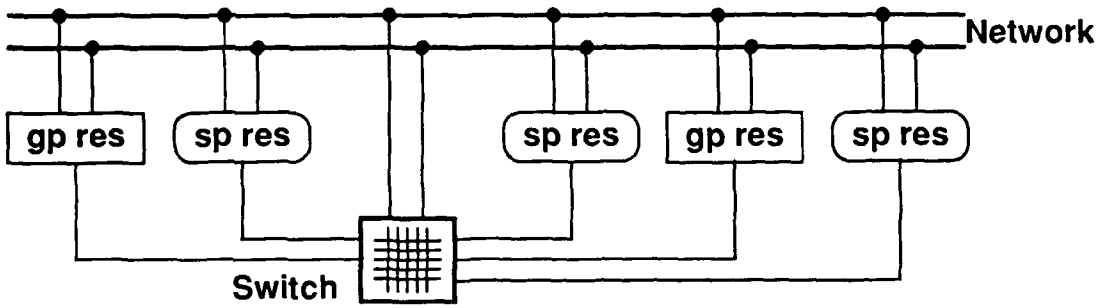


Figure 6. Resources layer.

Prototyping has shown a need for *inventoried boxes*, that are on the network and contain numbers and types of things (e.g., buoys, torpedoes, or missiles).

The Navy requires mission-specific or Navy-unique special-purpose resources, such as specialized communications and navigation equipment, sensors, and weapons. These can be simplified by unbundling the general-purpose components (e.g., by using a high-speed switch to separate the processor from the sensor, to allow connectivity when required, and to support sharing and exploitation of redundancy). However, special-purpose resources are not generally available off-the-shelf. To assure that they can be integrated into the host systems when delivered, the resources must be built according to the interface specifications of the architecture. Each must connect into SRRS via a RIP.

The resource layer is a leverage point for the government with respect to resource cost and ease of system configuration. Interface standards at this level, between resource BIT and the RIP, will constitute guidance to resource builders. That is, to be in compliance with SRRS (and combat system) guidelines, a resource must comply with the the BIT-RIP interface. This interface has yet to be defined, but information across it must be sufficient to support RIP reports as described previously (under applications).

OPERATIONAL EXAMPLE

Figure 7 shows an example of how layering applies to an operational function. At the functions layer, *locate submarines* is requested by a user. This function is decomposed at the services layer into *deploy sonobuoys* and *process sonobuoy* data. Those two services are partly implemented at the applications layer. For example, a *signal analysis* application is needed to process data. At the utilities layer, the *file manager* installs applications and utilities where they are needed. In the example, the *file manager* loads the *signal-analysis* application into a *signal processor*. At the same layer, the

resource manager first verifies that resources needed to perform the services are available and then allocates resources as appropriate. At the operating-system layer, messages are used to allow status checking and resource allocation by the resource manager and to transfer the *signal-analysis* application from the file manager to the *signal-processor*. At the resource level, a *signal processor*, an *interface signal-switching unit*, and a *radio receiver* implement the *process-sonobuoy-data* service.

In general, resources may be dedicated, scheduled, or shared. In this example, resources are dedicated to the process-sonobuoy service, to be deallocated when the respective service sessions are complete. Note that not all processes and resources are shown in figure 7.

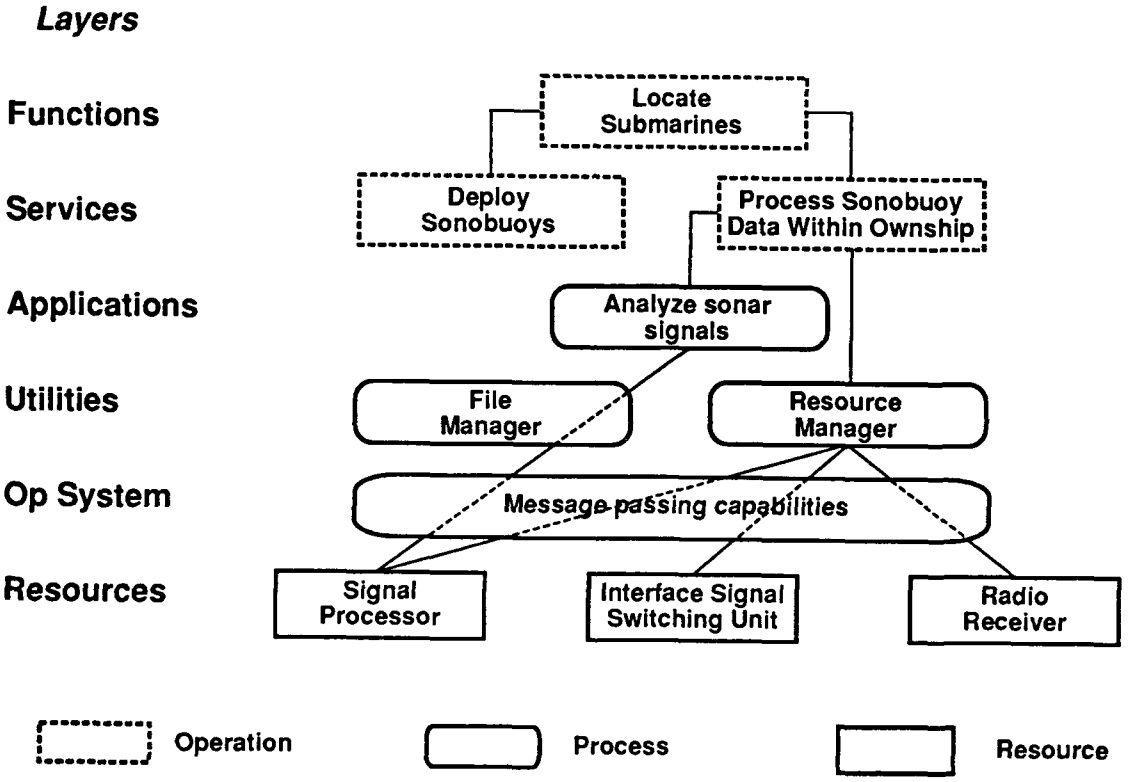


Figure 7. First layering example — locate submarines.

Figure 8 shows an example of how layering applies to a readiness function. At the functions layer, *report readiness* is requested by a user. This function is decomposed at the services layer into *report operational capability* and *report maintenance requirements*. Those two services are partly implemented at the applications layer. *Report operational capability* is partly implemented via the *tactical-readiness process*; *report maintenance requirements* is partly implemented via the *tactical-readiness process*. These two applications receive reports from the *resource interface processes*. At the utilities layer, the *file manager* installs the *tactical-* and *technical-readiness* processes on available workstations;

resource interface processes are installed on the respective resources. The *resource manager* also receives those reports it uses to check and allocate resources. At the operating-system layer, messages are used to interconnect the applications and utilities. At the resource level, *workstations* are used to host the *tactical-* and *technical-readiness processes.*

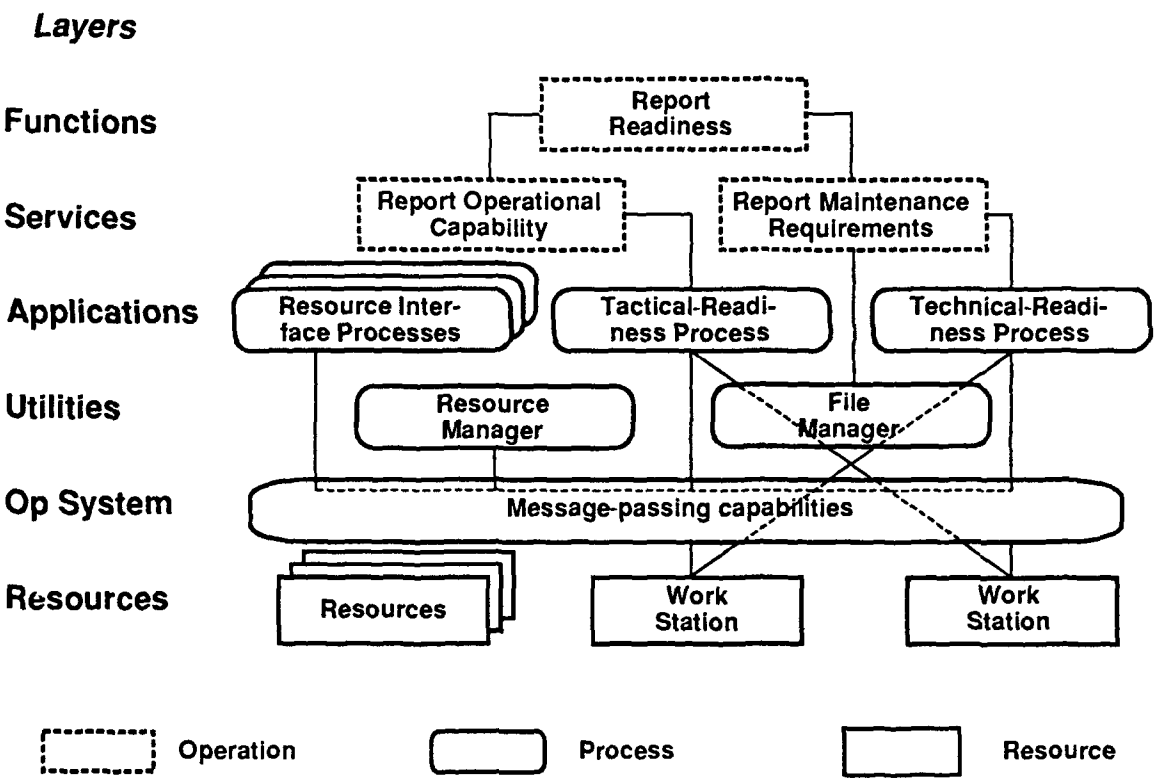


Figure 8. Second layering example—report readiness.

SOFTWARE PROTOTYPE

The SRRS software prototype is being developed to expose and confront issues inherent in designing a multiuser, priority-driven SRRS that is also a distributed, survivable, hierarchical, service-based, shared-resource system.

The software prototype is written in MODSIM II (reference 6) (a product of CACI Products Co.). MODSIM II is an object-oriented, discrete-event simulation language based on MODULA II (reference 7), linked to a color-graphics package. The software prototype is hosted on a SPARCstation 2 (a product of Sun Microsystems).

The word "system" will be used to refer to the set of hardware resources and software processes resident on the platform. The purpose of the system is to perform functions for the user on command.

MULTIPLE-BUILD APPROACH

A "two-build" approach is taken to develop the software prototype. Build 1, the FY-92 prototype, accepts scripted input. User interaction with the operating prototype is limited to obtaining graphical information concerning the simulation state. All events—user logon, requests for service, and damage—must be specified before initiating a prototype session and cannot be altered during that session. HM&E system operation is omitted from Build 1.

Build 2—the planned FY-93 prototype—will allow a user at the host workstation to interact with the prototype to log on, boot the system up, request functions, and read the resulting SRRS reports. In addition, the user can cause damage (i.e., to inject faults or to destroy resources) and evaluate the general SRRS response as well as the system reconfiguration. HM&E system operation will be included in Build 2.

The effort required to merge graphical interaction with the model of SRRS operation is expected to be tractable because the process has been broken into two builds. Because introducing a user into the process will yield new information, it should also be valuable.

ORGANIZATION

Under Build 1, the software prototype is organized into these parts: a *parameter file* is a user-created specification of all initial system-configuration information; an *event file* is a user-created specification of events—startups, user logons or logouts, function entries, or damage—and respective event times; a *main module* drives the prototype by first reading in the parameter and event files to initialize the prototype and then starts

a prototype run; and various program *modules* embody SRRS. Under Build 2, the event file will be eliminated in favor of user interaction.

Each program module contains two parts: a *definition part* specifies variables, procedures, and objects in that module and the calling conventions for the procedures and object methods; an *implementation part* contains the code that is executed when procedures and object methods are called.

OPERATION

To operate the prototype under Build 1, the user simply enters the command "srrs" from the host SPARCstation keyboard. The main module initializes the prototype from the parameter and event files—and then starts it. Graphical aids are available to observe execution progress, however, the user cannot intervene to change the outcome of the run. Under Build 2, a graphical interface will support user intervention.

In each build, the software prototype represents a distributed system. In general, software processes that must communicate with each other are on separate resources. A message-passing scheme is used to allow processes to communicate; messages trigger most object activity. The distinction between a primary and a backup copy of a process is as follows. Although both copies may create outgoing messages in response to an incoming message, a primary copy actually transmits outgoing messages, while a backup copy simply logs them for use in case the primary copy is destroyed. Later, the backup copy deletes those messages that match messages received from the primary copy. The SRRS message-passing scheme will eventually be mapped onto a *client-server model* that supports various qualities of service (e.g., assured delivery).

Input Files

Parameter File. Parameters are entered into the parameter file to specify

- User names and areas of responsibility
- Network names and tap numbers
- Network interface unit (NIU) names and network taps
- Resource names, numbers, and NIU connections
- Service names and required resources, applications, and utilities
- Element names and services
- Function names and supporting services
- Message priorities
- Prototype switches (to enable debug features).

Event File. Time-related events are specified in the event file. The simulation run-time limit and intervals are specified for reading from networks and writing to them, and updating status. These events can be specified:

1. Resource startup

2. Resource turnon
3. Operator logon (before a system boot)
4. System boot
5. User logon
6. User logout
7. User function entry
8. Resource damage.

Modules

Main Module. The main module reads the parameter file and enters the appropriate parameter values, reads the event file and loads the event queues (by telling various object methods when to execute), and starts the simulation.

Common Module. Certain objects are specified in the common module for use throughout the program. A *message object* has a type, a priority, an origin, and a destination. It also has a content queue that can contain zero or more content objects (data words). A *content object* can contain a string, a Boolean value, an integer, an object, or a combination of these. A *mailbox object* can contain a set of message objects, ordered by priority. A *subscriber object* has a name and a mailbox and can send and receive mail. A *driver object* has a timeout method that halts the simulation as soon as the run time has expired. A variety of types, variables, constants, and procedures is defined for common use. Common module objects are shown graphically in Appendix B.

Platform Module. A platform is the space within which all SRRS prototype activity takes place. All resources available to the prototype are on the platform at system-start time. No resources may be imported into the platform after system start.

User Module. A user is the commanding officer or a subordinate authorized by the commanding officer to issue commands to the system. A tactical user needs to know ship operational readiness. A technical user needs to determine maintenance requirements.

Within the user module, a *user object* is defined as a subscriber object (see common module) and inherits all subscriber object attributes and methods. Additional user attributes include a *host* (workstation), indicating if and where a user is logged on, and various trigger objects that cue the user to action. Key methods allow the user to bootstrap the system, log on and off, request functions, watch the workstation monitor for cues, and read mail. User module objects are shown graphically in Appendix B.

Function Module. A *function module* defines operational functions and readiness functions. Ship operational readiness is expressed in terms of the readiness to perform operational functions. Likewise, the maintenance requirements are based on what is needed to achieve a desired state of readiness to perform each operational function. Resource allocation is performed such that no function can cause a deadlock.

An array of function records, initialized by the main module as indicated in the parameter file, shows the services that support each function. A *function object*, needed for function instantiation, has attributes indicating the function type, priority, creator (a user), and a list of required services.

Three functions have been defined for Build 1: (1) *locate submarines*, (2) *report current platform position and velocity*, and (3) *report readiness*. The first two functions are operational functions; the third is a readiness function. In addition, a nil function is defined to allow a user to directly request a defined service. Function module objects are shown graphically in Appendix B.

Service Module. Within the *service module*, a *service object* is defined. Attributes of a service object include name, parent function, parent service, child services, and element. A check field is used to indicate whether or not a service can be performed. Service requests go to the responsible element controller. Service-module objects are shown graphically in Appendix B.

Application Module. Within the *application module*, a basic *application object* is defined as a subscriber object (see common module) and inherits all subscriber object attributes and methods. Key additional application object attributes include host (a resource).

A *resource interface object* (a RIP is of this class) is defined as an application object. Every resource that is not a workstation has a RIP whose primary duties are to receive mail for that resource and to report the resource status. Workstations have their own, more capable managers. RIP methods receive, analyze, and report the results of the resource built-in-test (BIT) to the TaRP and the TeRP.

A *workstation manager object* is defined as an application object. Every workstation has a manager whose primary duties are to receive mail for that workstation, report workstation status, log users on and off, load software processes (applications and utilities) onto the workstation on request, and to control the monitor (the interface between the user and the workstation).

A *tactical readiness object* (TaRP) and a *technical readiness object* (TeRP) are defined as application objects. TaRP and TeRP methods receive and synthesize reports from resource interfaces for use by the tactical and technical users respectively.

Application module objects are diagramed in Appendix B.

Utility Module. Within the *utility module*, a basic *utility object* is defined as a subscriber object (see common module) and inherits all subscriber object attributes and methods. Additional utility object attributes include a copy (all utilities will run with a primary and a backup copy), a station (a host workstation), a twin update time showing when the other copy last reported to it, and a message log to be used by a backup copy to recover from the loss of the primary copy.

A *file manager object* is defined as a utility object. The file manager has attributes describing active workstations and triggers used during system bootstrap. The file manager controls the system bootstrap, allocating primary and backup copies of utilities and certain applications to operational workstations. If possible, it also restores utility and application copies when necessary.

A *dbms object* is defined as a utility object. The database manager (a dbms object) has attributes detailing active users and workstations. It correlates user logons and logouts and updates mailboxes for users who are not logged on.

A *resource manager object* is defined as a utility object. In response to a service request from an element, the resource manager checks and reports resource availability. In response to an allocate request from an element, the resource manager allocates resources to a service at the given (service) priority and marks those resources unavailable. A *resource shadow object* has been defined to allow the resource manager to locally record and retrieve the status of each resource based on resource interface reports.

A *command and control interface object* includes a list of functions ordered according to priority. A function execution method is supported by methods to request service checks, resource allocation, service performance, and service preemption.

A *controller object* is defined as a utility object. Methods defined under this object allow an individual service to be checked, resources to be allocated, and a service to be preempted, in response to mail requests. *Element controller objects,* one for each of the five elements, are defined as controller objects. These objects have methods that perform the services unique to the element.

Resource Module. Within the *resource module* under Build 1, resources are characterized only to the extent required to determine availability.

A *network interface unit (NIU) object* is defined. Attributes include name, port (an object that in turn identifies a network and a tap), a send (outgoing) message queue, a resource, a user (for those instances when a user is logged on to a workstation connected to the respective NIU), and a list of mailboxes (belonging to subscribers on the resource). An NIU buffers mail for subscribers on the resource and receives mail from the network for those subscribers.

A *resource object* is defined. Attributes include name, interface, NIU, class, status, and condition (on, off, etc.). Resources can enter and monitor status and condition. A BIT method periodically checks for faults. Those that it finds (typically less than 100 percent of those present) are classified as redundant or nonredundant. BIT results are available to the resource RIP.

A *workstation object* is defined as a resource object. An additional method allows a workstation to load a file manager (in preparation for a system boot).

A *monitor object* is defined. Attributes include station and input and output buffers. Monitor methods allow it to act as the interface between a user and a workstation.

A *network object* is defined. Attributes include name, first and last tap, and a list of connected NIUs. The key network method supports transmission and reception of messages by NIUs.

A *buoy-box object* is defined as a resource object. A buoy box can keep an inventory of buoys.

Display Module. Under Build 1, two objects within the *display module* support graphical display of status information.

A *box object* can represent a function, service, resource, or logical gate (AND or OR) for display purposes. Attributes include name, position in the window, color, and type (used to determine the box shape).

A *graphic manager object* manages the graphic window. Attributes include window, a window object (provided by MODSIM II as part of the graphics package) and "boxes," an image object (also provided by MODSIM II as part of the graphics package) that in turn contains box objects. Methods allow the creation and display of a box, connection of one box to another, change of color (indicating condition) of a box, and deletion of a box. Another method allows the window to be reset in preparation for a new display.

IMPLEMENTATION APPROACH

GENERAL

This chapter proposes an approach for developing SRRS.

STANDARDS

The Next Generation Computer Resources (NGCR) program, directed by the Space and Naval Warfare Systems Command (SPAWAR 231), is defining standards for utilities and general-purpose resources. The NGCR program uses Navy-industry working groups to establish standards that are functionally useful and commercially viable. Sample NGCR standards are SAFENET (see below) and FutureBus (a computer backplane standard that will evolve from VME, a bus currently in wide commercial use). Other NGCR areas for standardization include graphics, operating systems, higher order languages, and high-speed data transfer networks.

As stated earlier, part of the SRRS philosophy is to specify what information resources must report rather than how to design those resources. This is the reason for establishing interface standards that satisfy both government and vendor needs. The NGCR program provides a good model for accomplishing this goal. Appendix C shows examples of specifications a vendor might work from to implement required reporting from resources.

Currently being studied is the subject of standards for reporting information to tactical and technical users. Prototype results in this area have potential value.

SAFENET TESTBED

The reasons for developing a software prototype are to characterize and document SRRS, debug the algorithms needed to make the SRRS work, and validate the SRRS concept. A natural next step in the prototyping process is to move SRRS to a hardware testbed.

The Survivable Adaptable Fiber-Optic Embedded Network (SAFENET) (reference 8) is a leading candidate for a standard shipboard local area network (LAN). SAFENET uses the Fiber-Optic Distributed-Data Interface (FDDI) protocol, a commercial standard. SAFENET development has attracted industry and triservice participation, a positive step toward joint and commercial dual use.

Two SAFENET testbeds are currently under development at NRaD. One of these will be chosen as the environment in which to refine SRRS concepts, once those concepts have been implemented, debugged, and shown to have merit in the software

prototype. The focus in this stage of development will be to make SRRS an integral tool for using and troubleshooting the testbed. SRRS will be refined to production quality to enhance—rather than degrade—testbed operation.

SUMMARY AND PLANS

This document has described an SRRS concept and an architectural framework, consisting of an open, service-oriented architecture that supports a hierarchical organization. Work remains to be done in defining functions and in decomposing them into services. In addition, standards must be developed for specifying resource interfaces and for producing user readiness reports. During the current fiscal year, a Build 1 software prototype has been developed characterizing an important subset of SRRS.

Future work will include developing a Build 2 software prototype and a hardware prototype: the former will improve upon Build 1 by adding an interactive capability that will allow users to log on, boot the system, request system functions, and access readiness reports; the latter will be based on the SAFENET LAN and will address issues of timing, performance, and extensibility.

REFERENCES

1. Vineberg, M., and S. Connors. 1991. "Shipboard Readiness Reporting System (SRRS)." NOSC TD 2154 (Aug). Naval Ocean Systems Center, San Diego, CA.
2. Naval Sea Systems Command, PMS-400. "Readiness Standards: Combat System Equipment Readiness Assessment and Reporting." S9410-AN-STD-010/AEGIS.
3. Nickerson, W. 1990. "System Study of Condition-Based Maintenance System for Shipboard High Pressure Air Compressors." DTRC-PASD.A-91 (Dec).
4. Vineberg, M. 1991. "A Proposed Navy Battle-Management Architecture," *BRG C2 Research Symposium*, National Defense University. June 1991, Washington DC.
5. "Portable Operating System Interface for Computer Environments (POSIX)," 1003.1-1988, IEEE, September 1988.
6. "MODSIM II, The Language for Object-Oriented Programming." CACI Products Company, La Jolla CA. January 1992.
7. Wirth, N. 1985. "Programming in MODULA-2." Third, corrected edition, Springer-Verlag, New York, NY.
8. "Survivable Adaptable Fiber-Optic Embedded Network II, SAFENET II. Military Handbook (MIL-HDBK-0036, Draft).

DEFINITIONS

Allocation — see resource allocation.

Application — a process that directly supports one or more services.

Architecture — a set of functions, the processes and resources that perform those functions, and the interrelationships among those functions, processes, and resources.

Backplane — a panel, generally located at the rear of an integrated-circuit-card enclosure, housing one or more (backplane) buses that support communication among the cards in the enclosure.

Backup — the passive copy of a process, identical to the *primary*, that communicates with the primary and operates so it can assume primary duties and request a new backup if the primary should be lost.

BIT — see built-in-test.

Built-in-test (BIT) — artifact (hardware and software) included within a resource to test and report the status of that resource.

Capability — the ability of a system resource to perform according to design specifications as derived from the *initialization*, *configuration*, and *failures* readiness categories and from such variables as expendables (ammunition, sonobuoys, etc.), the environment, and doctrine in use, qualified as follows:

up — available for use, as required, to all specified services without degradation; stores of expendables are at or near capacity; *failures* is *none* or *redundant, initialization* is *on, configuration* is *normal* or *alternate* (or possibly even the *casualty)*.

partial — can perform a subset of specified services; faults causing certain services to be partially supported or unsuported.

down — cannot perform as specified as a result of exhausted expendables — or various combinations of states in the *failures, initialization,* and *configuration* categories.

C2 interface — see command and control interface.

Central processing unit (CPU) — that part of a computer that performs data transformation.

Client-server model — a paradigm for network applications, well supported in the commercial sector, where a client (a process on one resource) may request service from a server (a process on another resource).

Combat system — the shipboard system that provides military attack or defense capabilities required by the ship's mission, including command and control, decision support, engagement, sensing, navigation, and communication.

Command and control interface (aka C2 interface, function controller) — an interface that enables a user to (1) request functions, (2) assign priorities to functions, (3) partition function repertoires into subsets, (4) delegate function subsets to subordinate users, and (5) delegate user priority-range subsets to subordinate users.

Communication element — the element that exchanges information with other nodes.

Configuration — the way a resource or system is interconnected and the applications and utilities are loaded; this category can be affected by *failures* and *initialization*, qualified as follows:

normal — up and fully integrated for the appropriate condition of readiness; the combat system may be in "Battle Short";

alternate — *a configuration* different from *normal* has been selected; capability may be slightly reduced, however, it is possible to move to *normal* (the combat system may be "Battle Short");

casualty — a fault has forced a move to this *configuration*, and capability may be reduced; corrective maintenance is required to enable a move to *normal* or *alternate*;

unavailable — broken and not configured for tactical use, off-line for maintenance or training, not fully initialized, or not authorized for use.

CPU — see central processing unit.

Database management — storing, updating, and retrieving related sets of data, generally accomplished by a database management system (DBMS).

Deadlock — a condition where, in a set of two or more services, each is halted, waiting for one or more resources controlled by one or more other members of the set.

Decision-support element — the element that supports command planning and decision making through assessment and evaluation and that accepts commands, provides system services, and returns responses.

Dependency tree — a tree including logical (AND and OR) nodes, illustrating the resources (represented by leaf nodes) required to perform a service (represented by an explanatory node).

Element — an entity defined according to a set of services it provides and that incorporates an element controller.

Element controller — an application within an element, active at all times, that handles interelement communication, service scheduling, and resource allocation and control during service sessions.

Engagement element — the element that destroys or neutralizes external threats.

Failures — the degree to which faults affect the ability of a resource or system to perform to specification, qualified as follows:

none — no faults; designed redundancy is intact;

redundant — faults affect redundancy only; loss of redundancy does not result in loss or reduction of any capability;

partial — faults prevent full performance of certain services;

total — faults prevent performance of all services.

File manager — a utility that stores and retrieves files, and transmits them on request, generally over a LAN, to a requesting device, such as a workstation.

Function — a system operation a user may request.

Function-based resource allocation policy — a scheme to prevent deadlock where a function cannot begin until complete resource control has been established by all elements concerned with all services needed to support the function.

General-purpose resource — a resource that supports a broad range of applications and utilities.

Graceful degradation — in response to damage, reconfiguration of a system to continue operation, although at a lower level of performance.

High-speed switch — a general-purpose resource capable of directly connecting two resources, e.g., a sensor or weapon and a processor.

Initialization — the degree to which a resource or system is turned on and its support services are available (can be affected by *failures* and *configuration*), qualified as follows:

on — the highest level of *initialization*; no further operator action is required to make the resource or system ready to respond to commands to operate;

standby — *energized*, available for use, all parameters are entered, and the necessary interfaces are made; further operator action is required to move to the *on* condition;

energized — all required power has been applied; operator action is required to move to the *standby* state.

support — power and all necessary support services are available; temperature, flow, pressure, etc., are within normal operating ranges;

off — not all required support services (power, air, cooling water, etc.) have been supplied.

LAN — see local area network.

LAN interface — the hardware and software that allows a device to connect to a local area network and use it.

Layered architecture — an architecture, partitioned "vertically" into layers, so that each layer supports the next higher layer.

Local area network (LAN) — a general-purpose resource, including one or more cables and the associated interfaces (hardware and software), allowing devices that observe interface specifications to connect (to the LAN) and to communicate (via the LAN).

Memory — that part of a computer that holds instructions and data directly accessible by the CPU.

Monitor — that part of a computer that includes a display screen and that is generally interactively controlled by a keyboard, a mouse, a touch panel, etc.

Navigation element — the element that plans, records, and controls platform position and velocity.

Open architecture — a layered architecture whose interfaces are defined by industry standards, controlled by recognized standards organizations (ISO, ANSI, IEEE, SAE, etc.).

Operating system — general-purpose software that enables the utilities and applications to use the hardware resources.

Operational capability — the ability of equipment systems to perform their intended functions, i.e., the material readiness; SRRS provides information about operational capabilities.

POSIX — IEEE Standard 1003.1-1988, Portable Operating System Interface for Computer Environments.

Preempt — withdraw resource access from one element and grant it to a second element to support a higher priority service.

Preemptable — characteristic of a resource that can be preempted if the priority of a newly requested service exceeds the priority of the service currently supported by the resource.

Primary — the active copy of a process, identical to the *backup*, that communicates with the backup and operates so it can request a new backup if that copy should be lost.

Priority — a characteristic of a command, supplied by the user, inherited by all supporting requests and services, and used to resolve contention for resources.

Priority range — a set of priorities, assigned to a user, that limits the priorities that user may attach to commands.

Process — software, capable of executing on one or more resources, designed to support one or more functions.

Readiness — the capability of the combat system, or portions of the combat system, to perform their intended functions when called upon to do so under prevailing tactical circumstances and environmental conditions; this refers to the capabilities of equipment and computer programs (material) and to the availability of tactically functional expendables and does not include consideration of factors relating to personnel readiness, logistic readiness, ship-engineering readiness (except as it pertains to provision of support services).

Readiness assessment — the collecting of data or information about the condition and state of equipment, computer programs and expendables, and the processing of those data or that information to determine the extent to which the equipment and systems can perform their intended functions or missions.

Readiness control — those activities responsible for providing or restoring specific combat system capabilities through initialization, configuration, reconfiguration, or maintenance and repair procedures.

Readiness status — the services available from a system, described in terms of capability (i.e., "up," "partial," "down").

Reply – information (acknowledgment, "cannot comply," etc.) returned by an element controller in response to a request.

Request — an order for service placed on an element by the C2 interface or by an element for which the performing element can supply a corresponding service.

Resource — hardware, either special-purpose (sensor, weapon, etc.) or general-purpose (local area network, workstation, etc.), needed to support processes.

Resource allocation — the granting of resource access.

Resource interface process (RIP) — an application, residing on each resource, designed to (1) determine the status of the resource based on BIT reports and (2) to report that status to the TaRP and the TeRP.

Response — information (acknowledgment, "cannot comply," etc.) returned from the services layer (the decision support element) to the functions layer in response to a command.

RIP — see Resource Interface Process.

Sensing element — the element that detects and characterizes objects and characterizes the environment external to the node.

Service — the mechanism by which support is supplied by the system in response to a request to implement a function. One service type may be used to implement more than one function type.

Service session — the period during which an element performs a service and retains control of the necessary resources.

Shipboard Readiness Reporting System (SRRS) — a system that combines data processing, communications, BIT, and calibration hardware and software to provide useful, accurate, concise, and timely readiness status reports to tactical and technical users responsible for operating and maintaining shipboard systems; the reports include the readiness data and information needed to make correct and timely operational and maintenance decisions.

Signal processing — transforming an incoming signal through numeric processing to extract information (signal processing tends to be computation intensive).

Signal processor — a device designed to perform (computation-intensive) signal processing.

Special-purpose resource — a resource that directly supports particular element services.

SRRS — see Shipboard Readiness Reporting System.

Subordinate user — a person authorized by a user at a node to access the BMA via the C2 interface and who the user provides with a command repertoire and a priority range.

System status — the condition or state of a system or part of the system described in terms of failures, initialization, and configuration.

Tactical user — a user responsible for employing the combat system, or its elements or components, to accomplish the tactical purposes for which the system was installed; his decisions require considering readiness information in light of the immediate tactical situation.

Tactical readiness process (TaRP) — an application that gathers a picture of system operational status characterized by those services the system can perform, separately or in combination, based on the current or projected status of system resources.

TaRP — see tactical readiness process

Technical readiness process (TeRP) — an application that gathers a picture of system operational status and supports scheduling of maintenance actions.

Technical user — a user responsible for ensuring that the combat system will meet the needs of the *tactical users* (1) for reconfiguring the system in an emergency to limit damage (according to previously adopted procedures) and (2) to coordinate required maintenance with the needs of the *tactical users*.

TeRP — see technical readiness process.

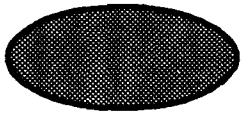
User — a commanding officer of a node or a subordinate authorized by the commanding officer to interact with the BMA via the C2 interface.

Utility — a process that supports a broad range of applications.

Workstation — a device (used primarily for data processing) — that includes a backplane, a memory, a CPU, a monitor, and a LAN interface.

APPENDIX A
SERVICE PROFILES

The following dependency trees use this symbology:



Logical node

Explanatory node

Leaf node

The highest level of each tree is an explanatory node representing the service. The leaves of the tree represent resource requirements. When a service is requested, the respective dependency tree is evaluated in two major steps.

> First, each leaf node is examined: if at least one resource is available to meet the resource requirement, the node takes value TRUE; otherwise, the node takes value FALSE.
>
> Second, higher nodes are assigned values: an AND node takes value TRUE, if all direct descendants have value TRUE — otherwise, it takes value FALSE; an OR node takes value TRUE, if any direct descendant has value TRUE — otherwise it takes value FALSE; an explanatory node takes the value of its (only) descendant.

If the resulting value of the highest level explanatory node (the service) is TRUE, the service can be performed; otherwise, it cannot.

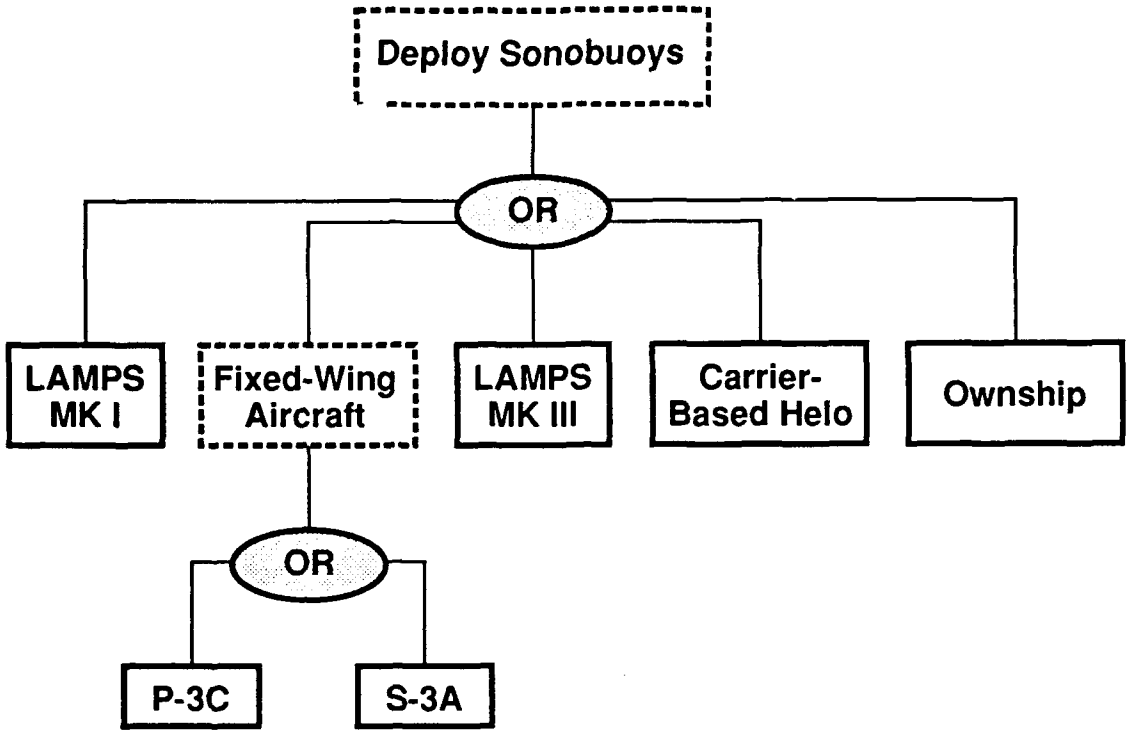
REPORT SYSTEM STATUS

At least one workstation is required (two are preferred) to run a TaRP and a TeRP.

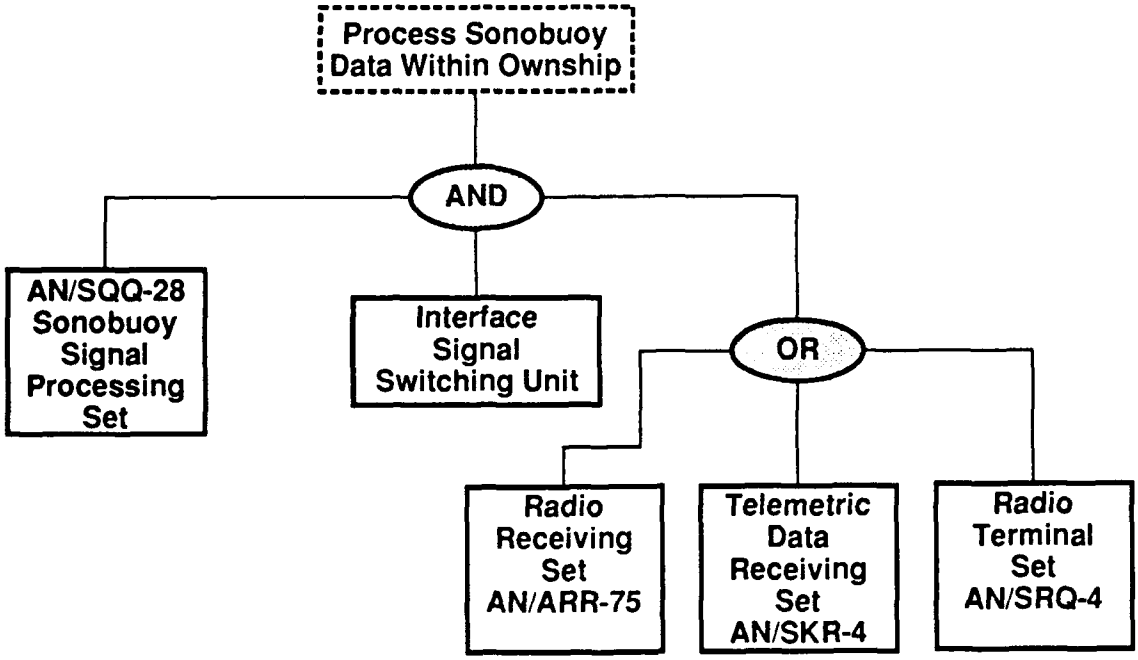
RECOMMEND RESOURCE CHANGE

At least one workstation is required (two are preferred) to run a TaRP and a TeRP.

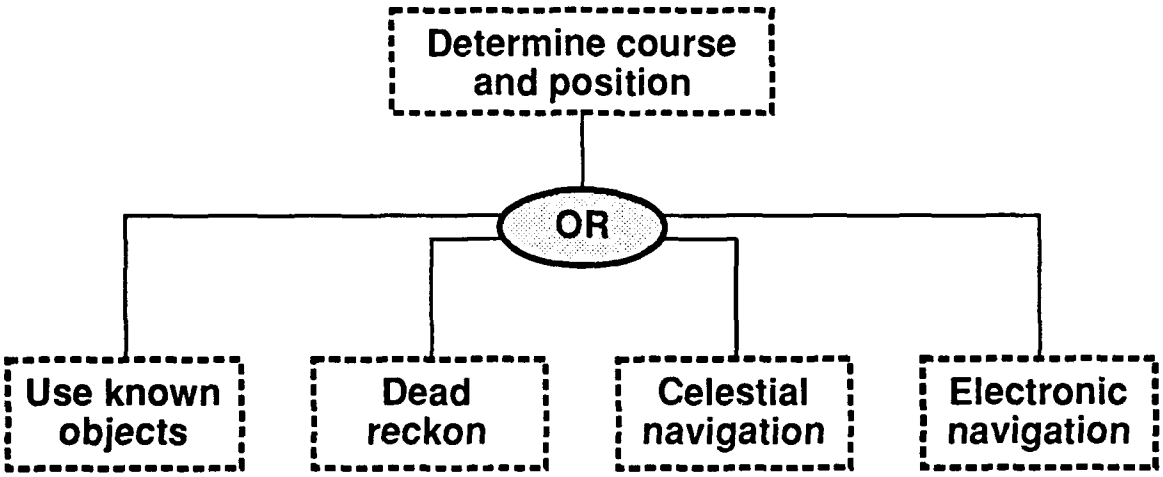
Deploy sonabuoys



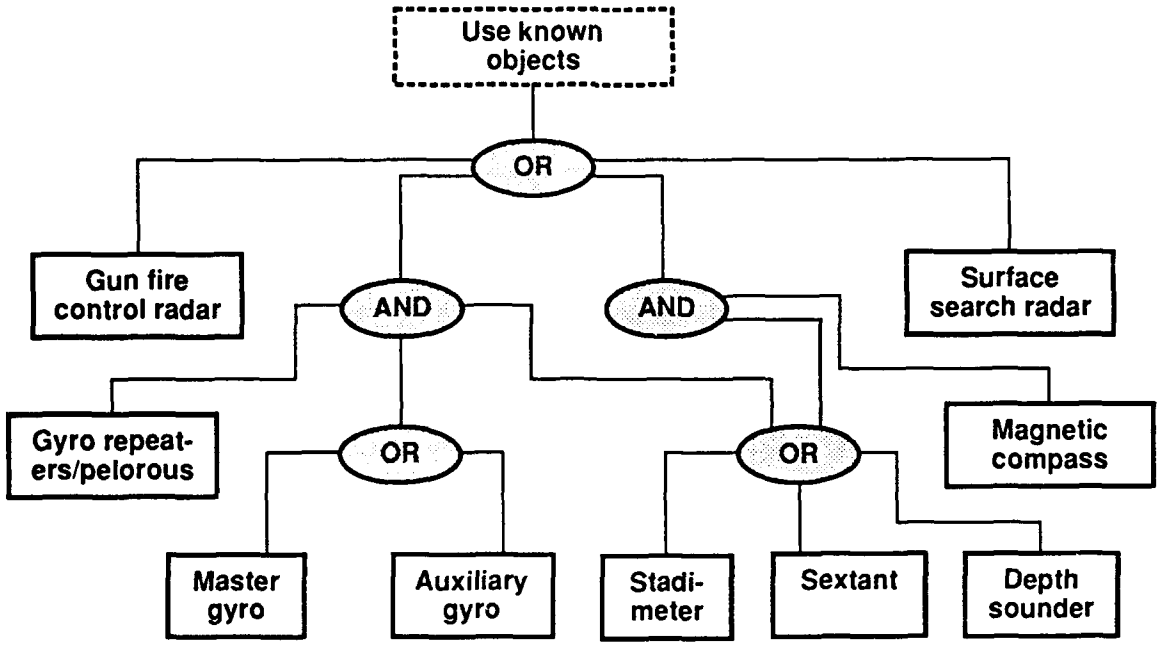
Process sonabuoy data

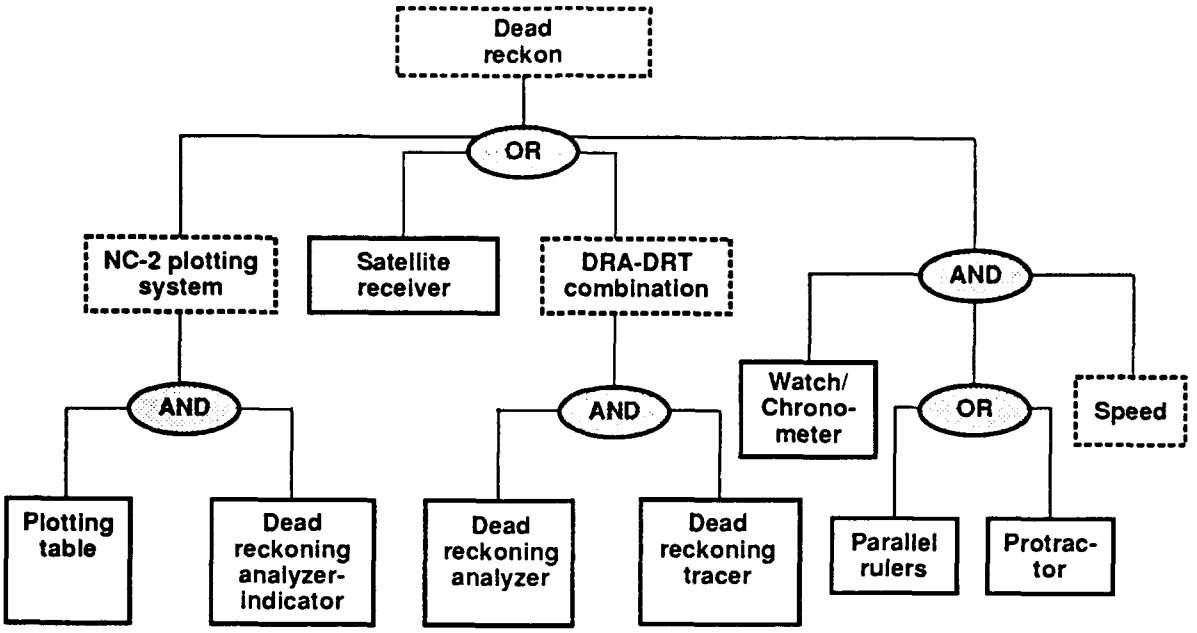


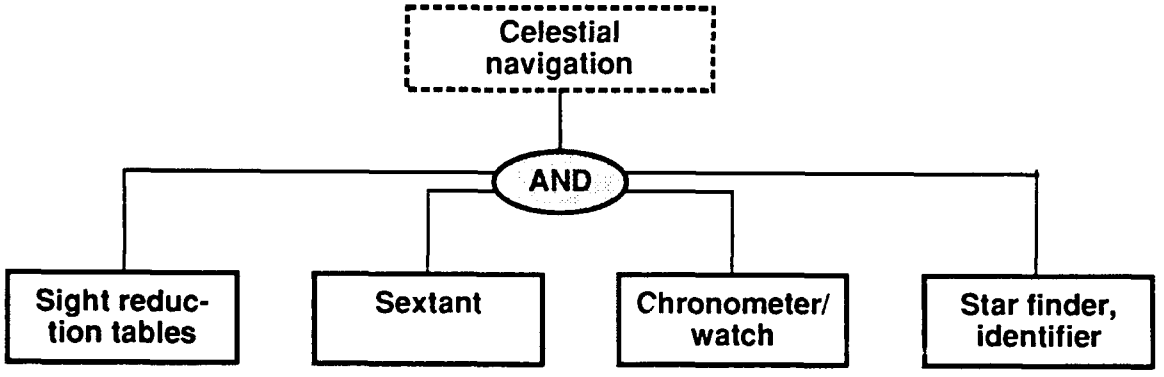
Determine platform course and position

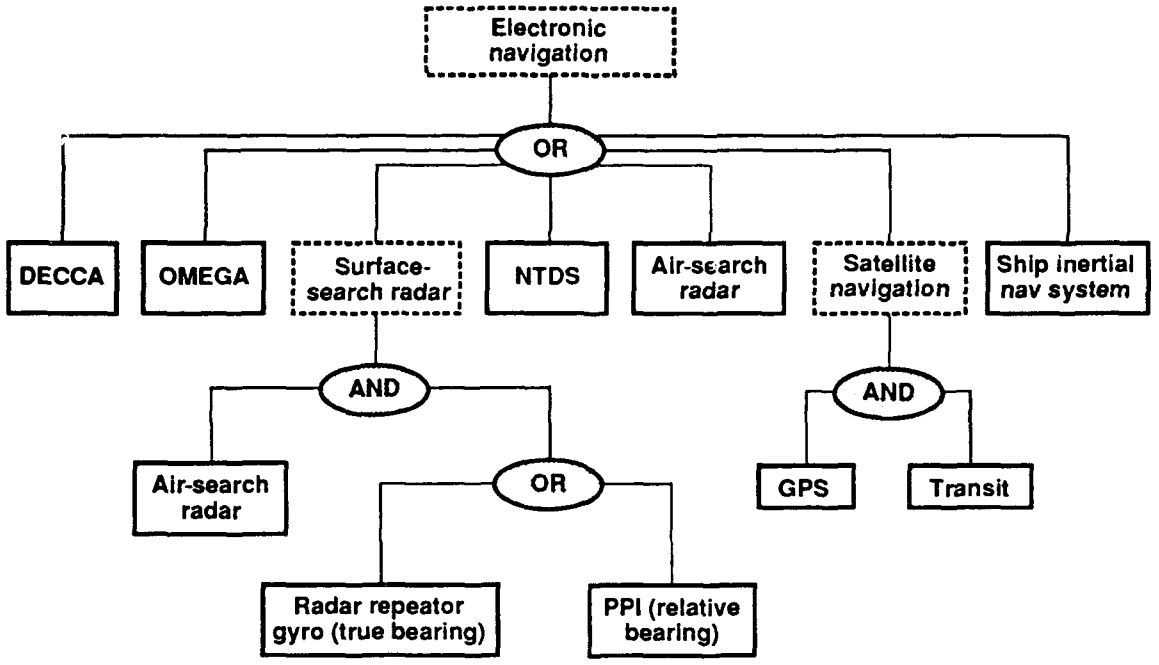


The explanatory boxes above are expanded below.

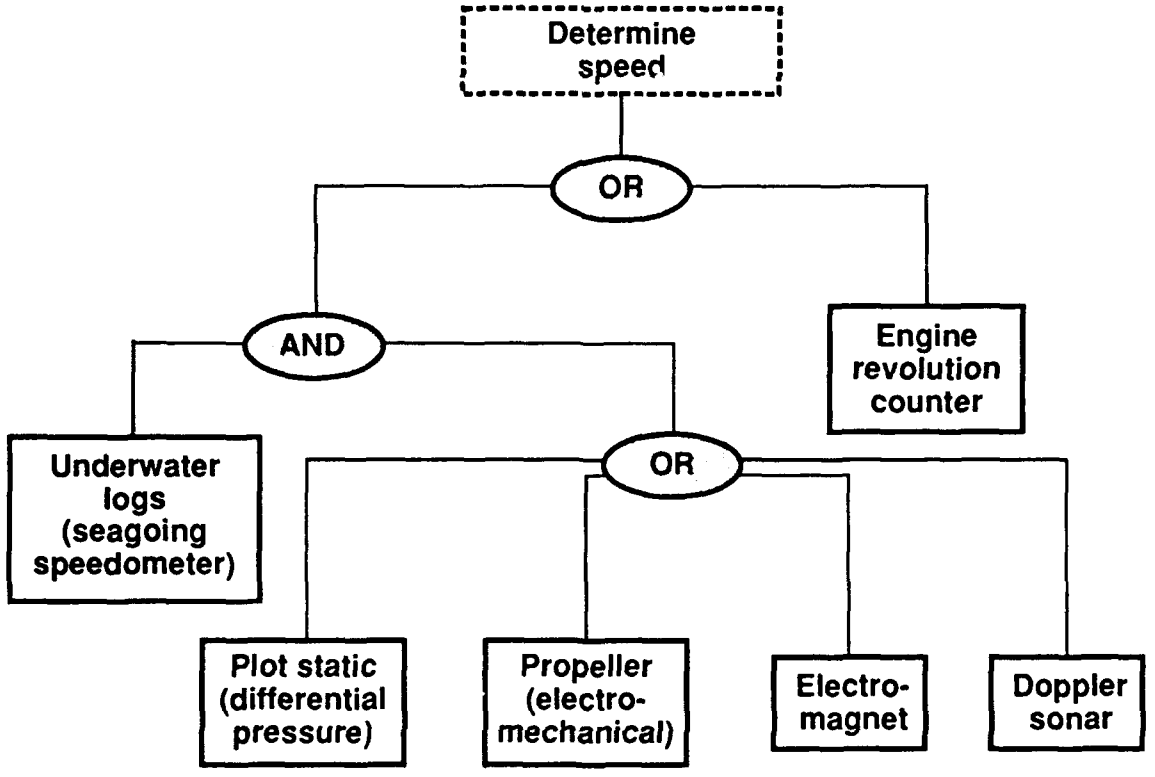



Dead reckon
OR
NC-2 plotting system
Satellite receiver
DRA-DRT combination
AND
AND
AND
Watch/ Chrono-meter
OR
Speed
Plotting table
Dead reckoning analyzer-indicator
Dead reckoning analyzer
Dead reckoning tracer
Parallel rulers
Protrac-tor


Celestial navigation
AND
Sight reduc-tion tables
Sextant
Chronometer/ watch
Star finder, identifier



Determine platform speed

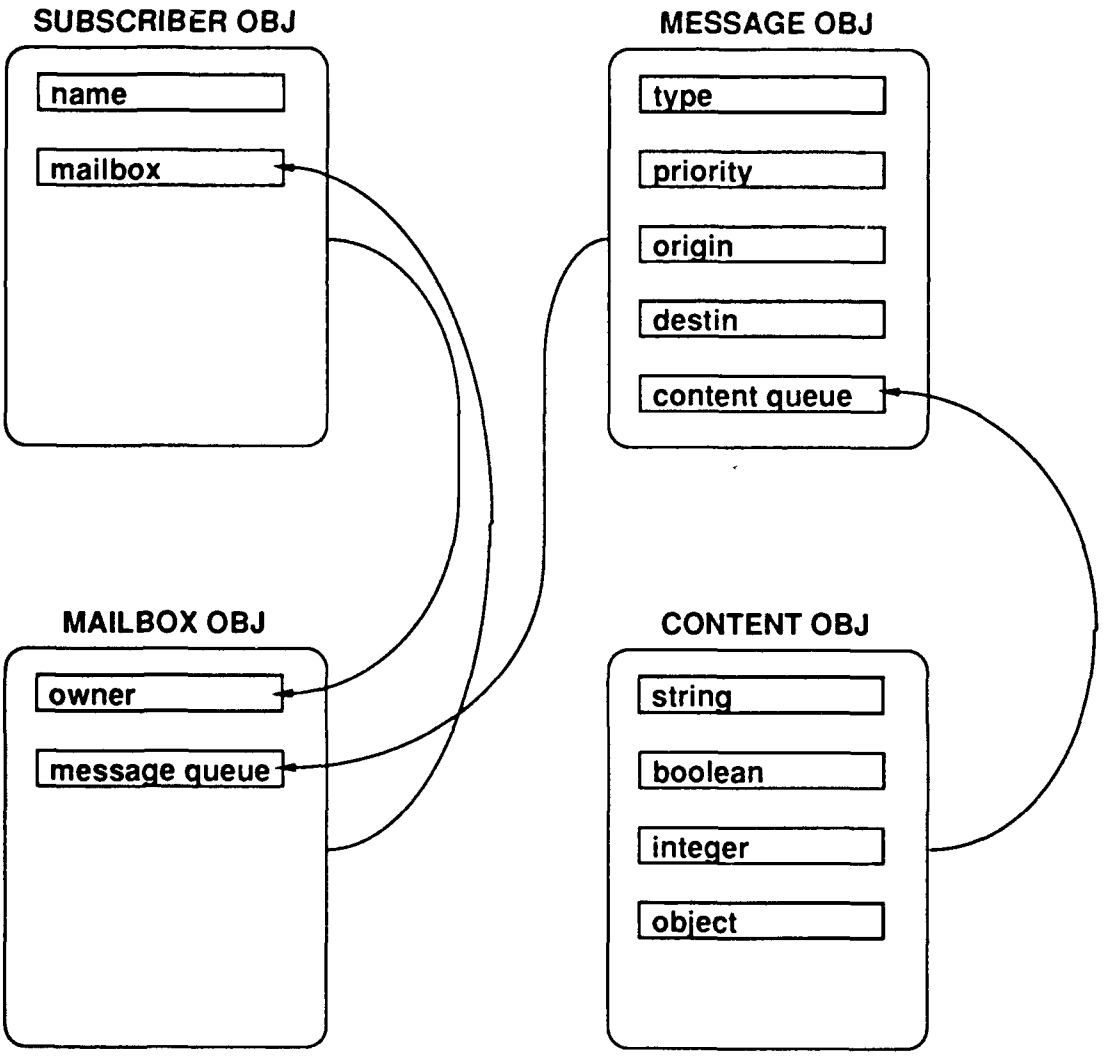


APPENDIX B
MODULES, OBJECTS, AND RELATIONSHIPS

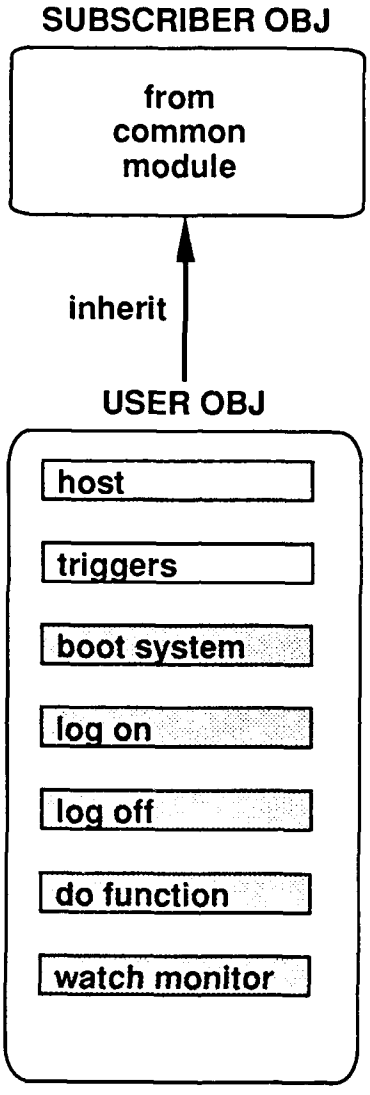
Important prototype modules and most of the objects in those modules are diagramed below. This is a pictorial representation of the relationships among parts of the prototype rather than a detailed programmer's reference guide.

Unshaded boxes within objects indicate attributes (data); shaded boxes refer to methods (operations). Arcs indicate the objects used as attributes either directly or as members of queues. In the diagram under the common module, a subscriber object may be a mailbox owner; a content object may be added to a message content queue.

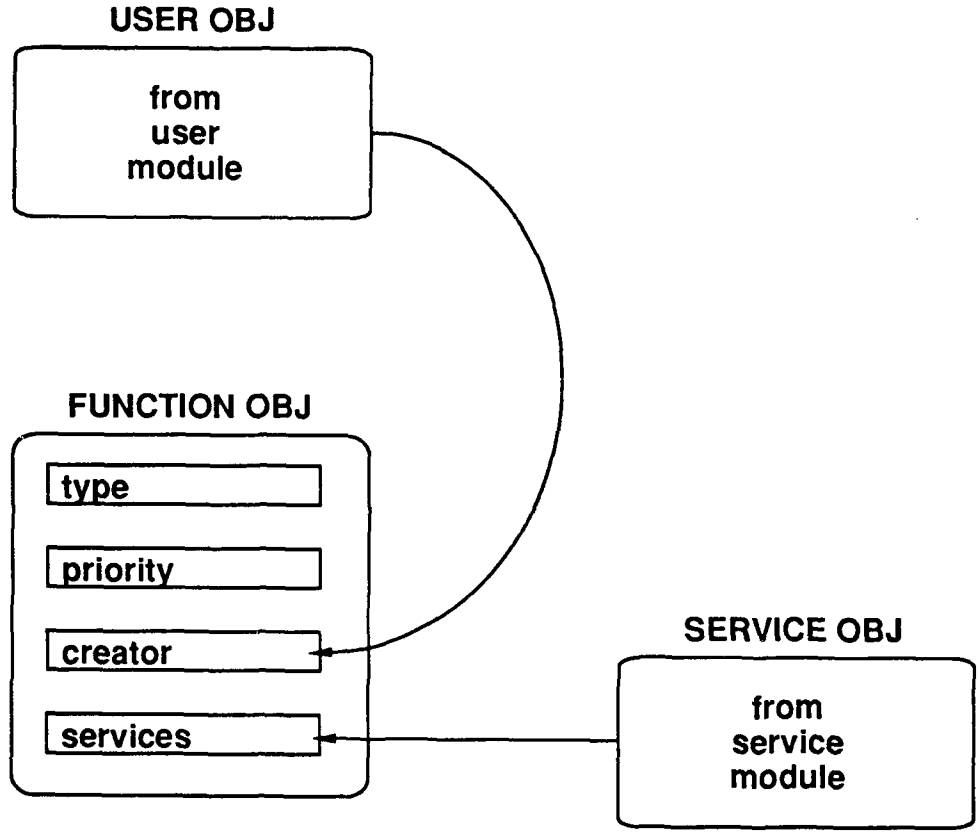
Common Module



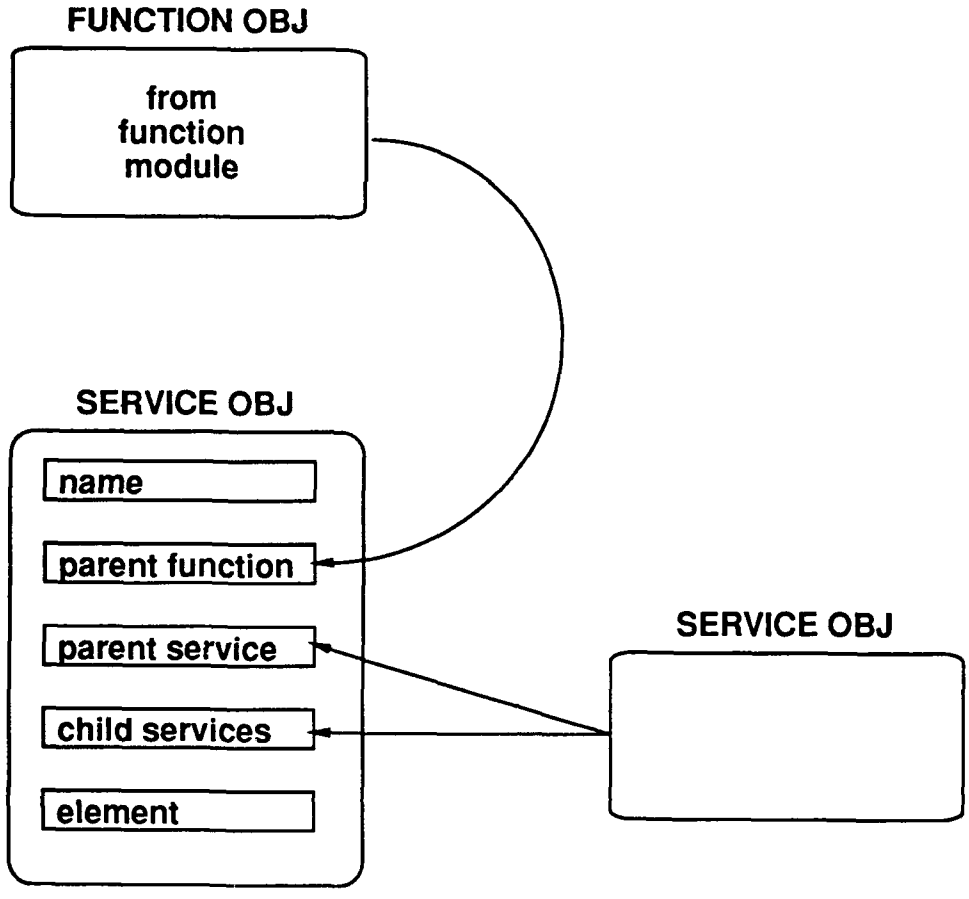
User Module


SUBSCRIBER OBJ
from common module
inherit
USER OBJ
host
triggers
boot system
log on
log off
do function
watch monitor

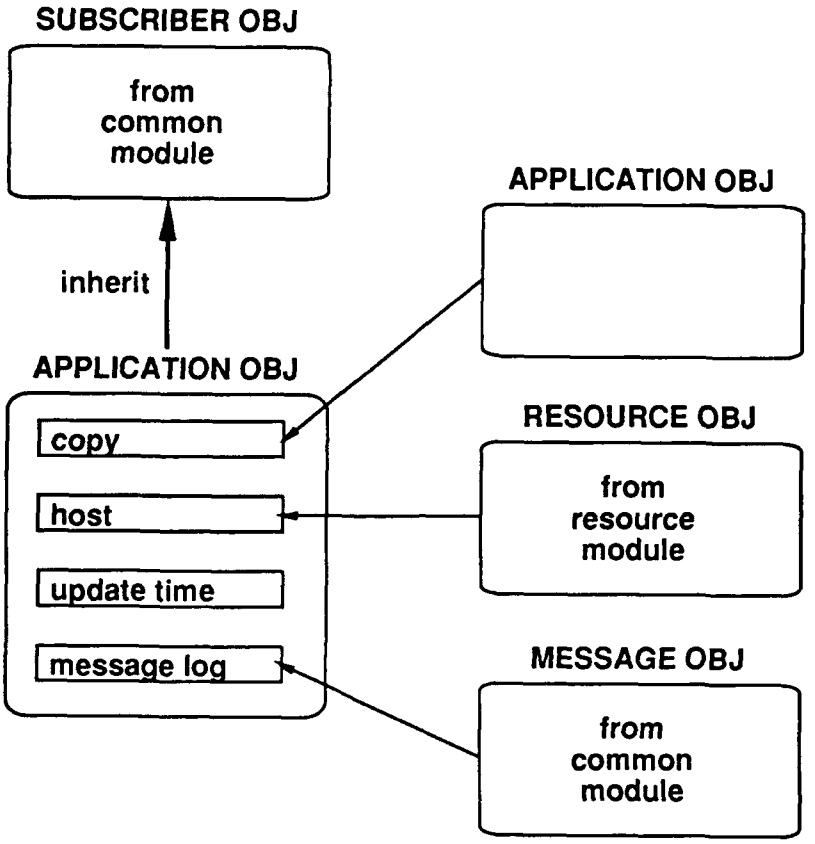
Function Module

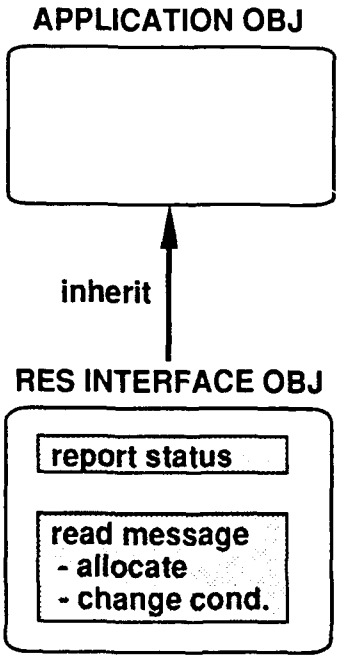


Service Module

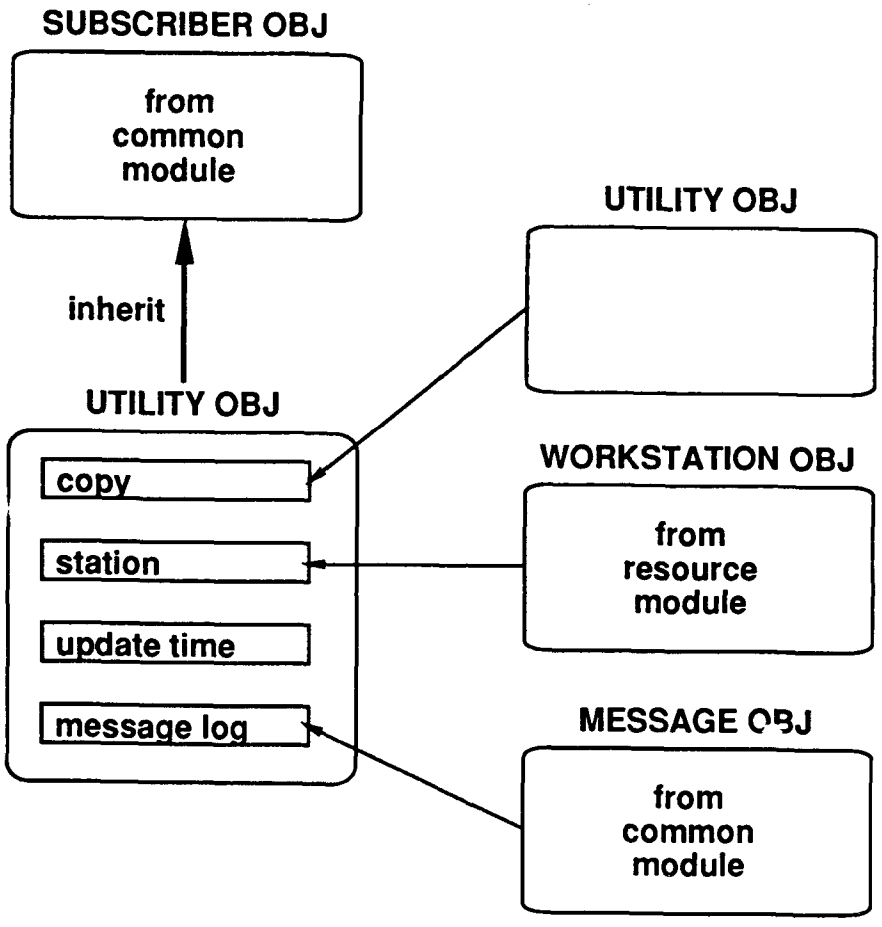


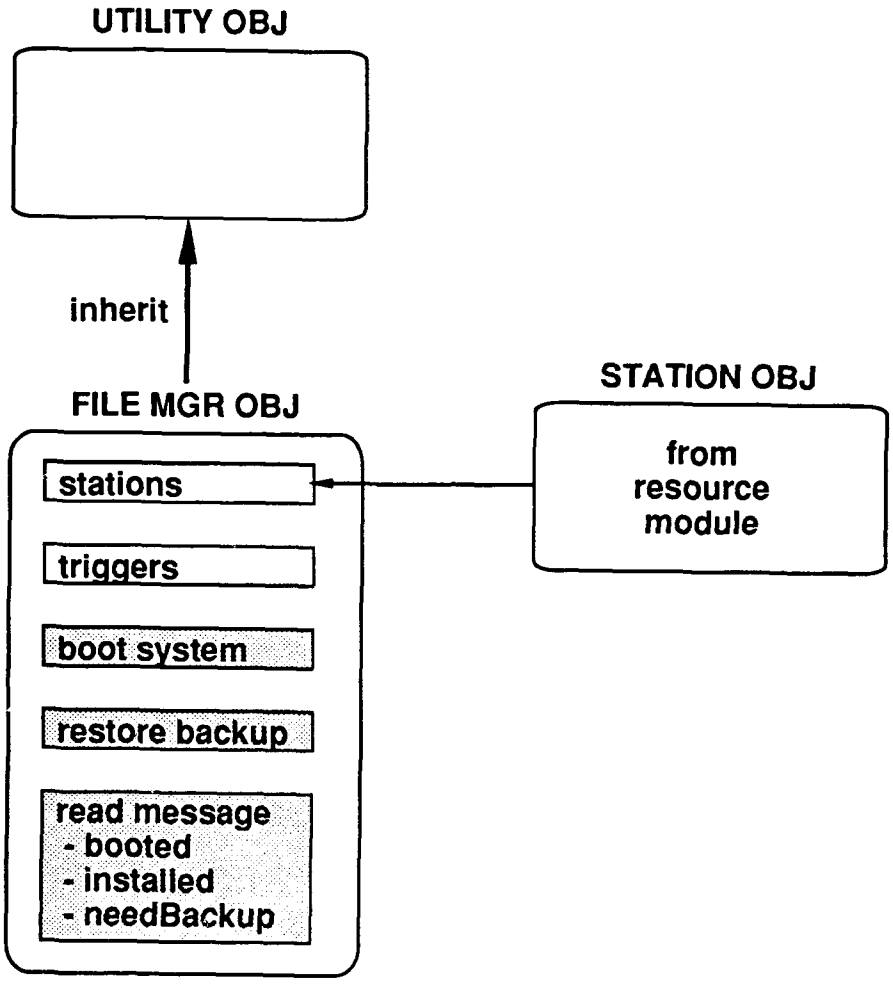
Application Module

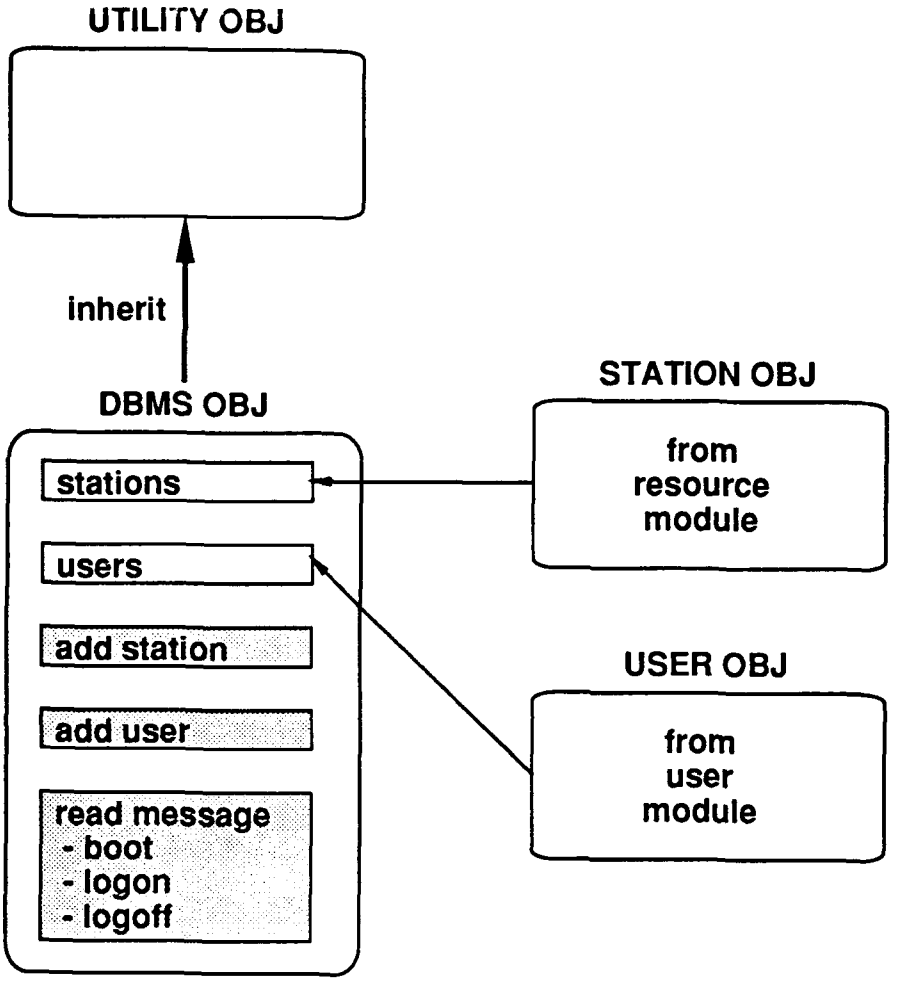


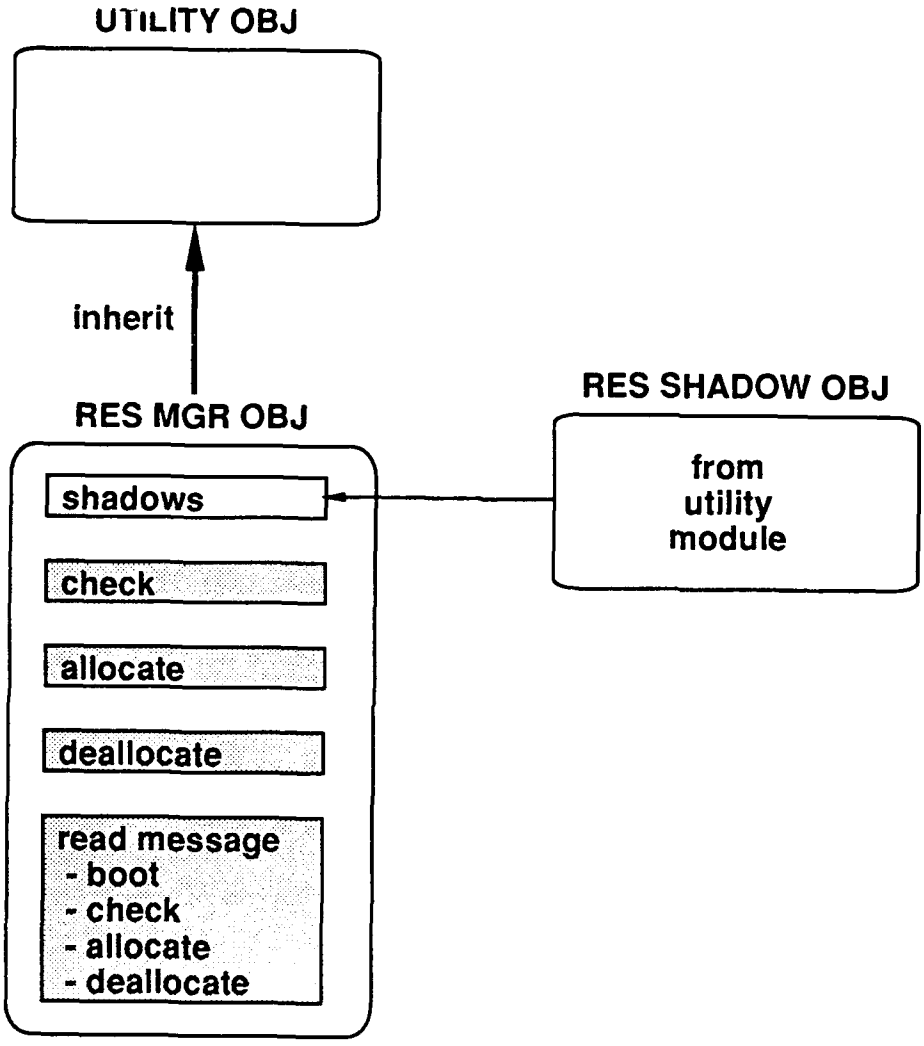


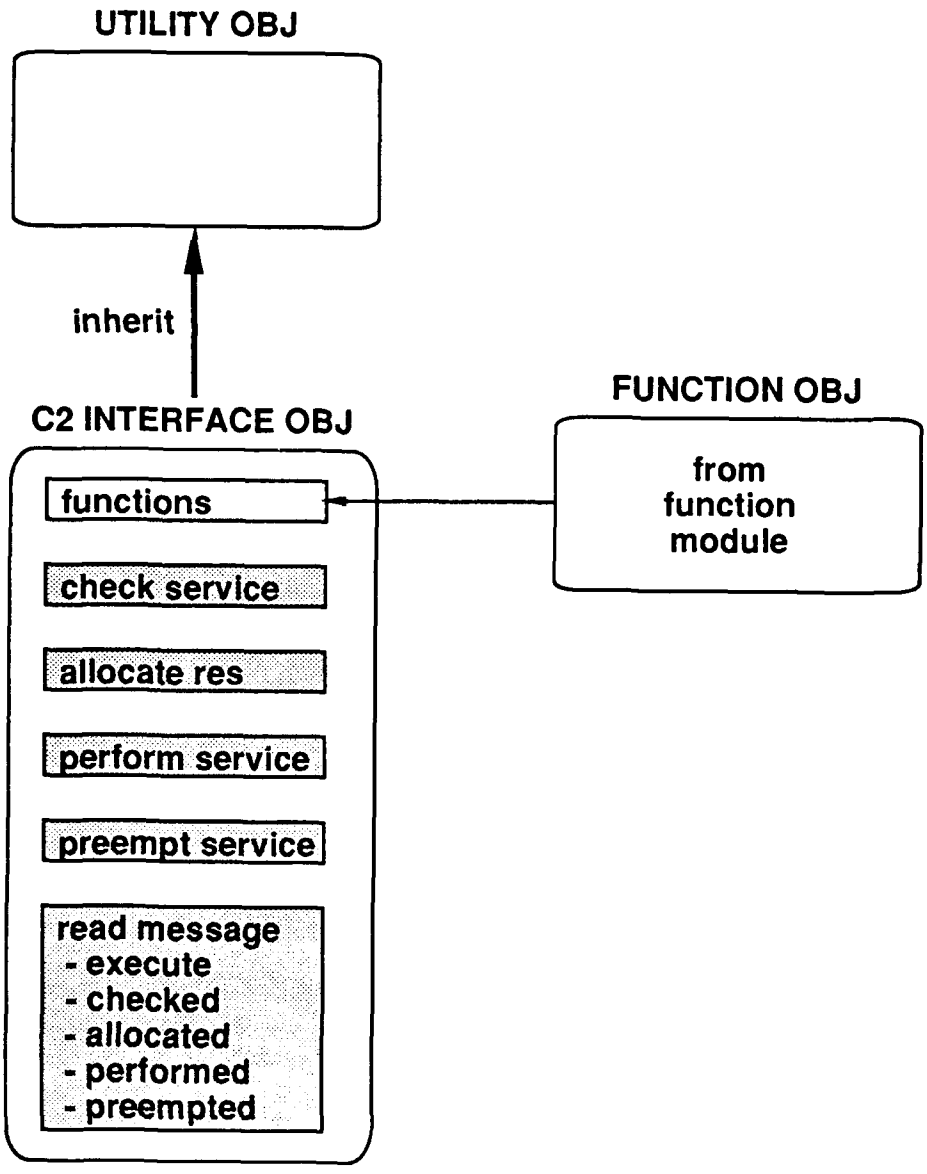
Utility Module

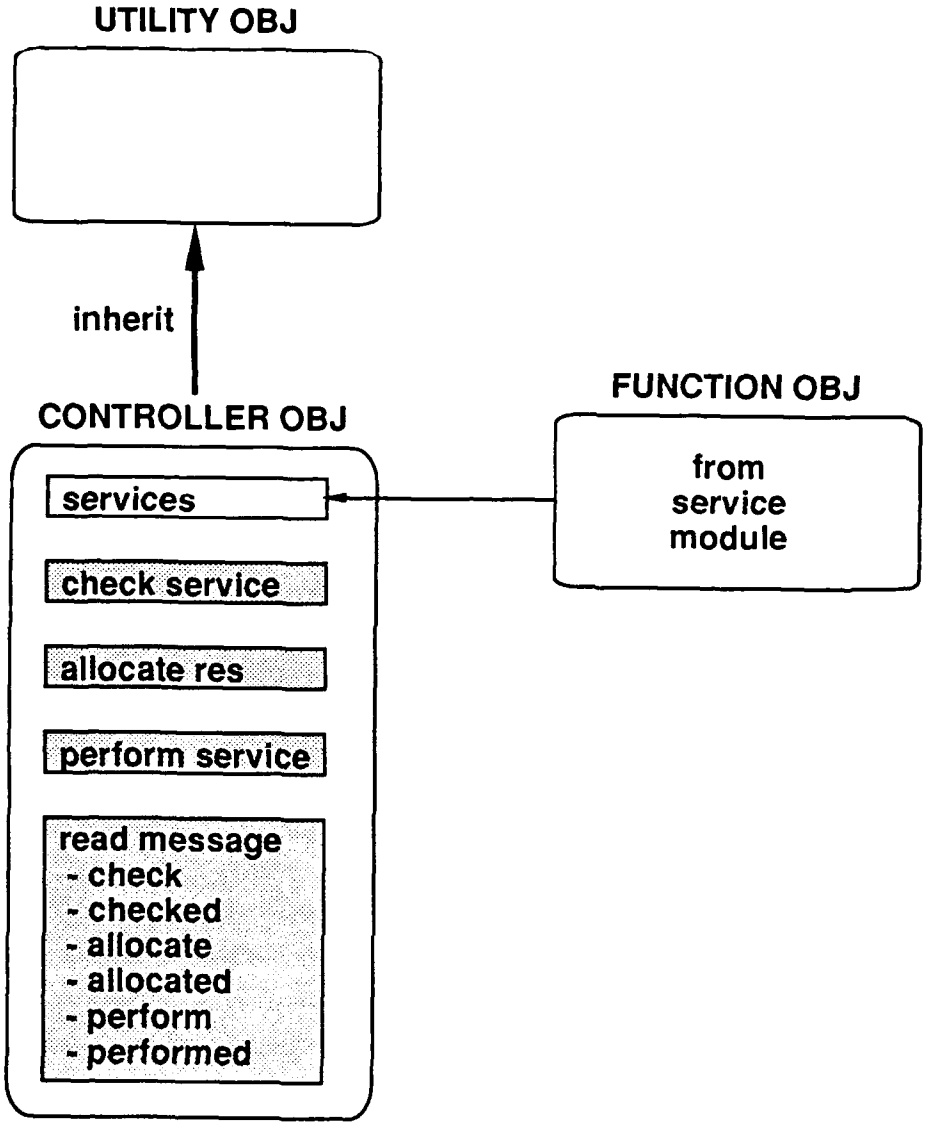



UTILITY OBJ
inherit
FILE MGR OBJ
stations
triggers
boot system
restore backup
read message
- booted
- installed
- needBackup
STATION OBJ
from
resource
module

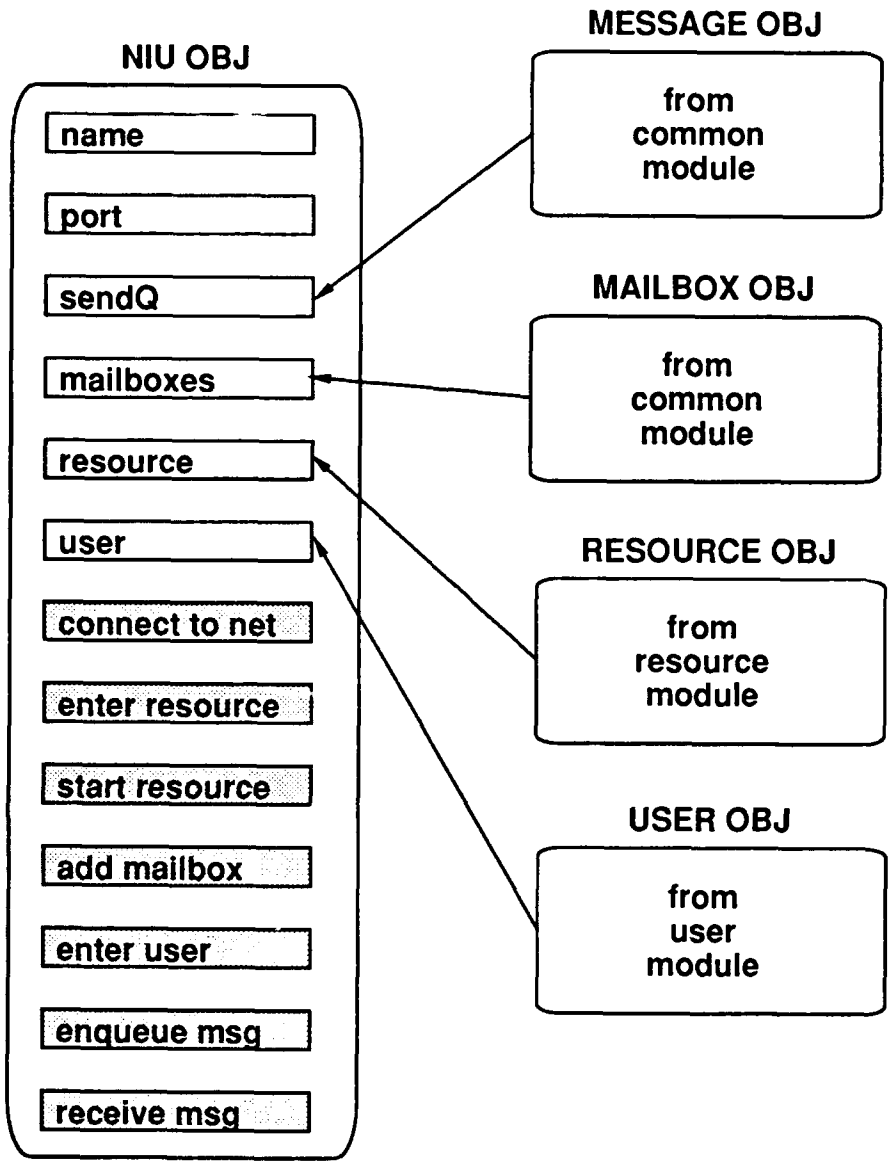

UTILITY OBJ
inherit
DBMS OBJ
stations
users
add station
add user
read message
- boot
- logon
- logoff
STATION OBJ
from
resource
module
USER OBJ
from
user
module

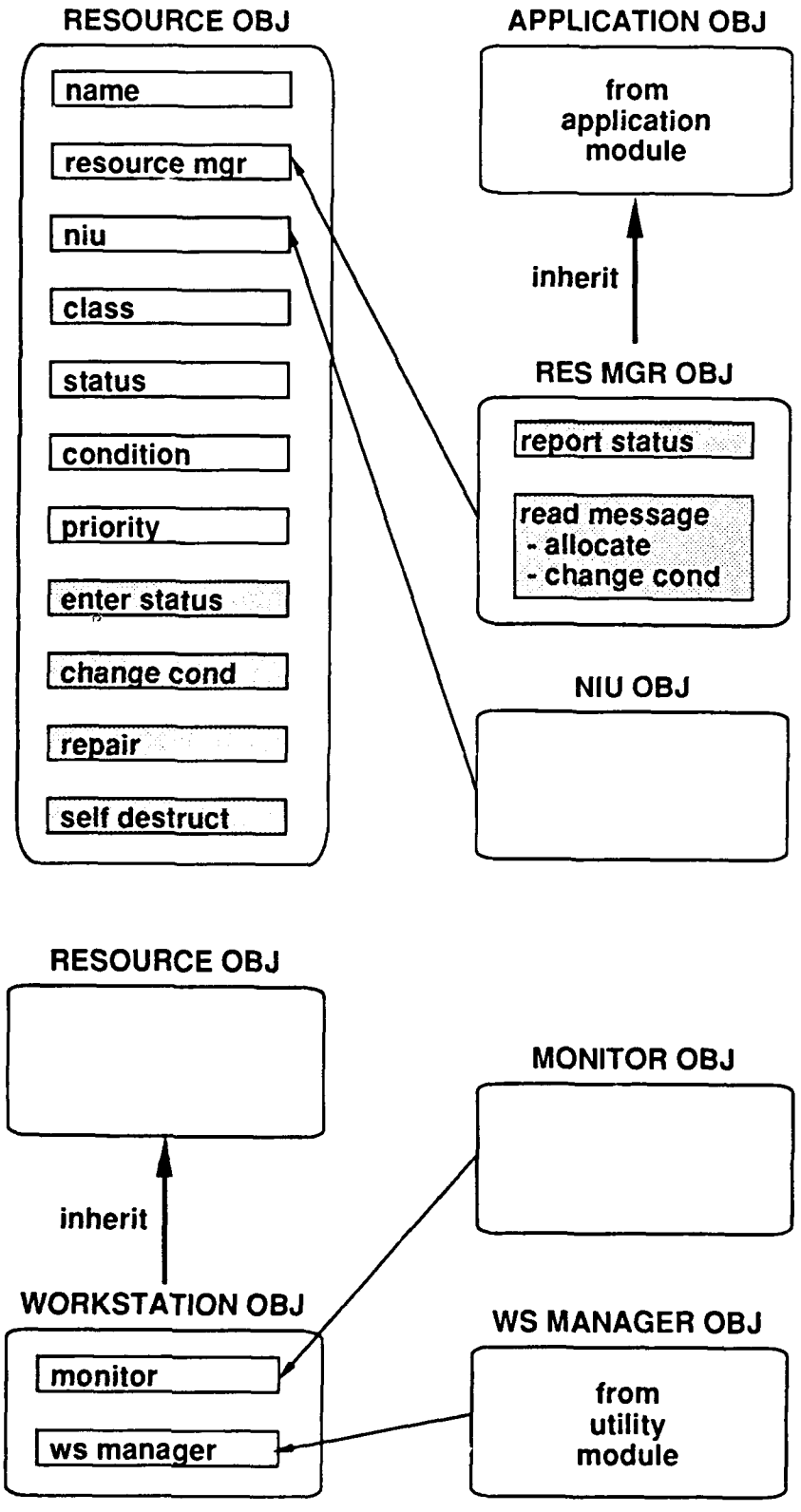

UTILITY OBJ
inherit
RES MGR OBJ
RES SHADOW OBJ
from
utility
module
shadows
check
allocate
deallocate
read message
- boot
- check
- allocate
- deallocate

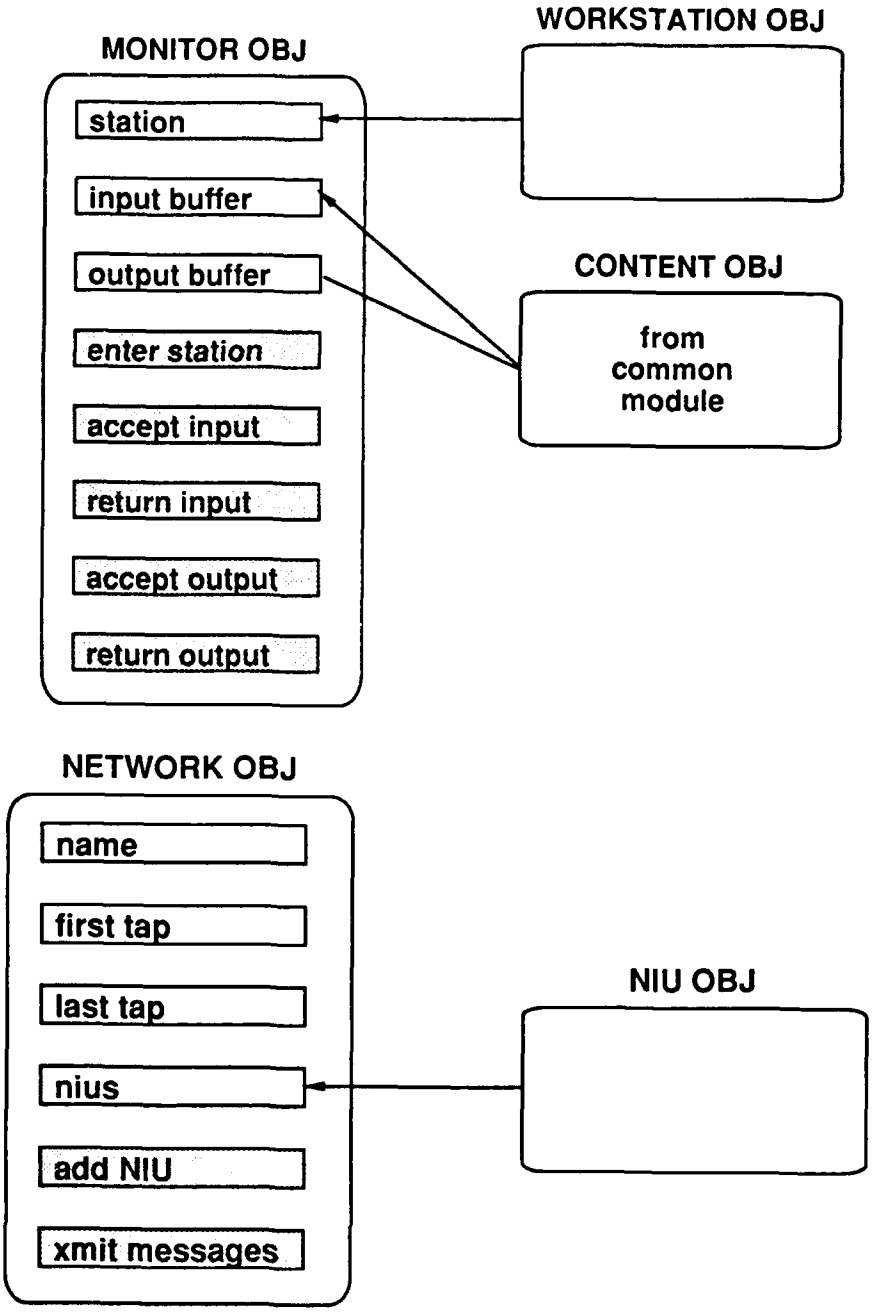

UTILITY OBJ
inherit
C2 INTERFACE OBJ
functions
check service
allocate res
perform service
preempt service
read message
- execute
- checked
- allocated
- performed
- preempted
FUNCTION OBJ
from
function
module

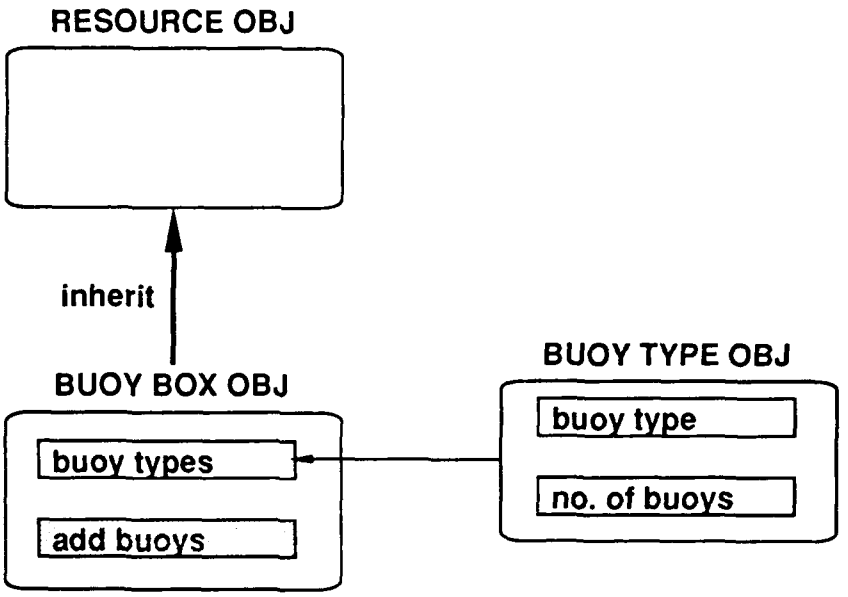

UTILITY OBJ
inherit
CONTROLLER OBJ
FUNCTION OBJ
services
check service
allocate res
perform service
read message
- check
- checked
- allocate
- allocated
- perform
- performed
from
service
module

Resource Module

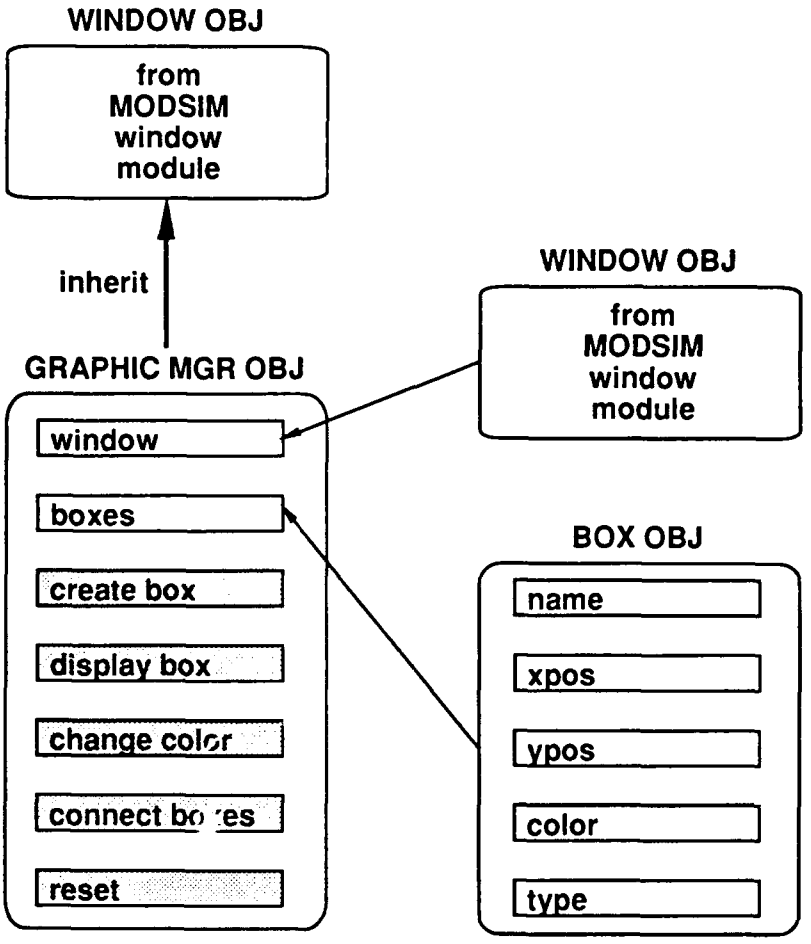



RESOURCE OBJ
name
resource mgr
niu
class
status
condition
priority
enter status
change cond
repair
self destruct
APPLICATION OBJ
from application module
inherit
RES MGR OBJ
report status
read message
- allocate
- change cond
NIU OBJ
RESOURCE OBJ
inherit
WORKSTATION OBJ
monitor
ws manager
MONITOR OBJ
WS MANAGER OBJ
from utility module


MONITOR OBJ
station
input buffer
output buffer
enter station
accept input
return input
accept output
return output
WORKSTATION OBJ
CONTENT OBJ
from
common
module
NETWORK OBJ
name
first tap
last tap
nius
add NIU
xmit messages
NIU OBJ


RESOURCE OBJ
inherit
BUOY BOX OBJ
buoy types
add buoys
BUOY TYPE OBJ
buoy type
no. of buoys

Display Module



APPENDIX C
DETERMINATION OF SHIP/SONOBUOY DATA-LINK STATUS

This is an example of how a designer could decide what means a particular resource had for reporting its readiness (and then how to build in that reporting capability). In other words, the example demonstrates how a designer could meet readiness requirements for reporting and assessment.

The three tables below show how readiness decisions are made for the AN/ARR-75, AN/SKR-4, and AN/SRQ-4 respectively. These tables differentiate among failures, initialization, and configuration and reveal how the lower level terms are used under each category.

AN/ARR-75

1. *Failures*
   - a. *none* – All 4 channels up
   - b. *redundant* – Not applicable
   - c. *minor* – 1 channel down
   - d. *major* – 2 or 3 channels down
   - e. *total* – All 4 channels down, power supply down, or power converter down.

2. *Initialization*
   - a. *on* – All power switches on (bulkhead, circuit breaker on the radio receiver group, and a toggle switch of the radio set control).
   - b. *standby* – Not applicable
   - c. *energized* – Not applicable
   - d. *support* – Bulkhead switch on.
   - e. *off* – Bulkhead switch off.

3. *Configuration* – Not applicable

AN/SKR-4

1. *Failures*
   - a. *none*
   - b. *redundant*
   - c. *minor* – Any one channel down, discriminator not functioning, trigger failed, demultiplexer not functioning (active out, but passive OK), loss of voice.
   - d. *major* – Loss of 12-dB attenuator, loss of post amplifier.
   - e. *total* – Either or both 28-volt power supplies, loss of preamplifier.

2. *Initialization*
   - a. on – All power switches on (bulkhead and front panels of SKR-4A), preamp of channel to selected helo on.
   - b. *standby* – Preamp not on, but all power on at receiver and bulkhead.
   - c. *energized* – Not applicable
   - f. *support* – Bulkhead switch on, switchboard set up properly.
   - e. *off* – Bulkhead switch off.

3. *Configuration*
   - a. *normal*
   - b. *alternate* – Test underway, test equipment set up, recorder connected.
   - c. *casualty* – Not applicable
   - d. *unavailable*

AN/SRQ-4

1. *Failures*
   - a. *none* – Uplink and downlink both working
   - b. *redundant* – Not applicable
   - c. *minor* – Loss of voice communications with Lamps MK-III Helo
   - d. *major* – Loss of uplink
   - e. *total* – Loss of downlink, over temperature conditions

2. *Initialization*
   - a. *on* – Bit cycle activated and running (approximately 2 minutes to complete).

b. *standby* - Not applicable

c. *energized* - AN/SRQ-4 turned on (power applied); bit cycle not activated.

d. *support* - Bulkhead switch on but nothing at AN/SRQ-4 is on.

e. *off* - Bulkhead switch is at off.

3. *Configuration*

a. *normal* - Transmit-receive/receive switch at transmit-receive

b. *alternate* - Not applicable

c. *casualty* - Directional antenna failed, omniantenna can be used (operational limitations).

d. *unavailable* - Built-in-test activated (not during initialization) or umbilical cord connected to Lamps MK-III Helo.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1992	Final FY 91 – FY 92

4. TITLE AND SUBTITLE
SHIPBOARD READINESS REPORTING SYSTEM (SRRS) SOFTWARE PROTOTYPE

5. FUNDING NUMBERS
PR: EE03
ACCESS NO: DN301014
PE: 0602233N
SUBPR: RM33D61

6. AUTHOR(S)
M. Vineberg, S. Connors, A. Sterrett, D. Shore

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division (NRaD)
San Diego, CA 92152–5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NRaD TD 2313

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
David Taylor Research Center
Bethesda, MD 20084–5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A software prototype of the Shipboard Readiness Reporting System (SSRS) is described. SRRS will ensure that the operational and maintenance readiness of the entire surface combatant is continuously measured and reported. It will collect and synthesize periodic status reports from shipboard resources and automatically report operational capability and maintenance requirements, based on resource status, to tactical and technical users. This system is designed in the context of a layered open architecture in anticipation of future shipboard architectures. The highest layer of the architecture includes both operational functions (relating directly to the ship's mission) and readiness-reporting functions.

14. SUBJECT TERMS
readiness reporting
local area network

15. NUMBER OF PAGES
62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(Include Area Code)*	21c. OFFICE SYMBOL
M. Vineberg	(619) 553–6557	Code 171

INITIAL DISTRIBUTION

Code 01	R. T. Shearer	(1)
Code 144	V. Ware	(1)
Code 17	J. Avery	(1)
Code 171	M. Vineberg	(30)
Code 405	C. Mirabile	(1)
Code 41	A. Justice	(1)
Code 411	A. Sterrett	(10)
Code 423	L. Peterson	(1)
Code 432	S. Connors	(5)
Code 432	T. Tiernan	(1)
Code 7502	R. Walker	(1)
Code 753	H. Quesnell	(1)
Code 82	P. Adams	(1)
Code 82	R. Kochanski	(1)
Code 821	R. Reed	(1)
Code 954	D. Shore	(5)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(2)

Office of Asst Secretary of Defense
Washington, DC 20301-8000

NCCOSC Washington Liaison Office
Washington, DC 20363-5100

Navy Acquisition, Research & Development Information Center (NARDIC)
Washington, DC 20360-5000

Naval Surface Warfare Center
Bethesda, MD 20084-5000

Naval Sea Systems Command
Washington, DC 20362-5101

Naval Ship Systems Engineering Station
Philadelphis, PA 19112-5083

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

Center for Naval Analyses
Alexandria, VA 22302-0268

Office of Naval Technology
Arlington, VA 22217-5000

Naval Surface Warfare Center
Annapolis, MD 21402-5067 (5)

Space & Naval Warfare Systems Command
Washington, DC 20363-5100

Naval Postgraduate School
Monterey, CA 93943-5000 (3)

Systems Exploration
San Diego, CA 92117 (2)